A Data Programming CS1 Course

Ruth E. Anderson, Michael D. Ernst
University of Washington
Seattle, WA
{rea, mernst}@cse.uw.edu

Robert Ordóñez
Southern Adventist University
Collegedale, TN
rordonez@southern.edu

Paul Pham
The Evergreen State College
Olympia, WA
phamp@evergreen.edu

Ben Tribelhorn
Seattle University
Seattle, WA
tribelhb@seattleu.edu

ABSTRACT

This paper reports on our experience teaching introductory programming by means of real-world data analysis. We have found that students can be motivated to learn programming and computer science concepts in order to analyze DNA, predict the outcome of elections, detect fraudulent data, suggest friends in a social network, determine the authorship of documents, and more. The approach is more than just a collection of "nifty assignments"; rather, it affects the choice of topics and pedagogy.

This paper describes how our approach has been used at four diverse colleges and universities to teach CS majors and non-majors alike. It outlines the types of assignments, which are based on problems from science, engineering, business, and the humanities. Finally, it offers advice for anyone trying to integrate the approach into their own institution.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—Computer science education

General Terms

Design, Experimentation, Human Factors.

Keywords

CS1; introductory computing; data programming; data processing.

1. INTRODUCTION

Traditional introductory programming courses often take their examples and assignments from the domains of puzzles, games, and abstract mathematics. For instance, students might be shown how to reverse a list or assigned to compute the Fibonacci sequence. This approach works well for some students, but others may fail to see how the concepts being taught can be applied to the world beyond computing. This lack of connection to the real world can have a negative impact on students' motivation and interest in CS.

Recently there has been an explosion in the number of datasets available from every conceivable field, and "Big Data" is in the news everywhere [16]. Graduate students find they need to take programming courses to solve problems in domains such as biology, physics, economics, and the social sciences [e.g. 27].

We have developed a CS1 course with a "data programming" orientation that teaches students introductory programming and computing concepts by means of real-world data analysis problems. The course is intended for undergraduate students with no prior programming experience. The materials were originally developed at the University of Washington where the course has been taught three times, and they have been adapted and used at multiple other institutions. This paper reports on our experience at a diverse set of four of these schools.

The key idea of our approach is that assignments use an existing dataset to answer a question that is relevant to science, engineering, business, or the humanities. Neither the questions nor the datasets are artificial. We have successfully used this approach to

- motivate and excite students about computation,
- teach computer programming & computer science concepts,
- enable students to process data to solve real problems of interest to them, and
- serve both CS and non-CS majors.

Although the approach is not necessarily tied to any one language, the offerings we report on have been in the Python programming language. Our goal in this paper is not to formally evaluate these offerings but rather to describe the courses, show the variety of settings where the approach has been applied, and enable other instructors to reuse the ideas and/or materials in new settings.

2. RELATED WORK

Many educators have suggested alternatives to the puzzles-and-games approach to CS1. Several have noted the benefits of placing CS concepts in contexts students find motivating [5]. Well-known examples include media computation [20, 26], robotics [15], and animation [6]. Stevenson's real-world programming assignments [23, 24] are a web crawler, spam evaluator, and steganography. De-Pasquale [7] presents three data sources: stock quotes and the APIs from Google and Slashdot. These domains are mostly connected to entertainment and computers, whereas our approach shows computation applied to science, engineering, business, and the humanities.

Other educators have sought to show students how the skills they are learning can have impact on the world beyond computing. The ITiCSE working group [12] offers 14 projects related to "social good" that motivate students and provide them with skills to solve complex problems; Erkan et al. [11] do the same for a data structures course. By contrast, we weave the theme of solving problems with real-world impact throughout the course. Some courses that address realistic problems and data focus on visualization [21], statistics [3], or databases [25]. Ours is a CS1 course that teaches computational thinking and programming.

Others have sought to create CS1 courses that demonstrate the applicability of CS to STEM fields [2, 29]. Many, though not all [18], of these courses use the Python language as we do. Dodds et

SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.
Copyright © 2015 ACM 978-1-4503-2966-8/15/03…$15.00.
http://dx.doi.org/10.1145/2676723.2677309

al. created a breadth-first CS1 course for scientists [8]. Hambrusch et al. used problems from scientific domains to teach computational thinking [14]. Our course teaches practical programming skills, and our strong focus on real datasets has an impact on our pedagogy, course topics, and assignment structure.

3. OUR INSTITUTIONS

We have successfully implemented this approach in four different courses taught by five different instructors at our four institutions.

3.1 University of Washington (UW)

UW is a public, R1 university with approximately 40,000 students. UW has a thriving two-quarter CS1 and CS2 sequence in Java that is a prerequisite for admission to the CS major. Entrance to the major is competitive and while there is no CS minor at UW, there are eight non-majors CS courses available on topics ranging from web programming to artificial intelligence.

The goal of the data programming course at UW is to offer an alternative to students who are not attracted to the CS major or to the Java CS1 offering. CSE 160 Data Programming has been offered three times (Summer 2012, Winter 2013, Winter 2014) by two different instructors to classes of approximately 50 students. UW is on a ten-week quarter system, and students in CSE 160 attended three 50-minute lectures and one 50-minute recitation section per week.

CS majors are not permitted to register for CSE 160. Students from a variety of other majors have taken the course, and some have gone on to double-major in CS as well as their original field (e.g. Bioengineering, Oceanography). Students have multiple times ranked the course among the 10 best in the College of Engineering, and it always has a waiting list.

3.2 Pacific Union College (PUC)

Pacific Union College is a private, liberal arts parochial four-year college with approximately 1,700 students.

The data programming approach was used in CPTR 115 Introduction to Computer Programming during two consecutive quarters (Fall 2012 and Winter 2013). This course is a prerequisite for a "breadth-first CS1" [4] course for computer science majors (usually taught in C) that assumes prior programming knowledge. It is also a cognate (requirement outside the major department) and usually the only programming course for mathematics, biomathematics, engineering, and physics majors, as well as an option in the Practical and Applied Arts area of the general education requirements at PUC. Thus it is a course required of CS majors and taken by students from several other majors simultaneously. It met for three 50-minute lectures and one 3-hour lab period per week during a ten-week quarter.

The two offerings of CPTR 115 were taught by the same instructor to 16 and 8 students, respectively. Roughly half of those students were declared CS majors. A handful were high school students in a dual-enrollment program, two of whom went on to become CS majors the following year. Focusing on real-world data sets in a variety of fields made this course far more engaging for its broad audience, compared to previous offerings. Starting this focus from the very beginning of the term enabled students to tackle a sizable open-ended data analysis project within the confines of a 10-week quarter. Anecdotal feedback from students indicated a higher-than-usual level of enthusiasm for the kinds of problems tackled and a sense of accomplishment at doing real data analysis — even for those who had previously done a bit of game programming.

3.3 The Evergreen State College

The Evergreen State College is a public, liberal arts teaching college with approximately 4,400 students. Evergreen is known for its unique interdisciplinary approach. The demographic of Evergreen students includes a higher percentage of non-traditional students than those at an R1 university: older adult students, working and single parents, and veterans.

Data programming was first offered at Evergreen in Fall 2013. This course was the first (CS1) of a three-quarter sequence in introductory computer programming. The remaining two programming courses in the sequence were taught in Java and did not use a data programming approach. This sequence is combined with two other three-quarter sequences (in computer architecture and discrete mathematics) to form a lower-division CS curriculum (called a program), which is a prerequisite for an analogous three-quarter upper-division CS program. Although Evergreen does not have official majors and departments, students who take these two programs self-identify as CS majors. Evergreen is on a ten-week quarter system, and for this course students attended two 2-hour lectures and one 2-hour lab per week.

The 68 students in the CS1 course consisted of both majors (who intend to continue onto the upper-division CS program) and non-majors (students for whom this course will be their only exposure to CS). The CS1 course received positive reviews from students, students were highly engaged in the class, and the turnover rate to the follow-on Java CS2 course was low.

3.4 Seattle University (SU)

Seattle University is a private, masters-level university with about 7,300 students, mostly undergraduates.

CPSC 192 Data Driven Programming is a new course designed for non-CS majors and CS minors. It serves as the first course in the CS minor and as the CS requirement for the Environmental Science major as well as some tracks of other science majors (i.e. Math). Students majoring in CS at SU take a two-quarter sequence in C++. All CS courses except the data programming course are closed to non-majors, so demand for this course is expected to remain high as entry into the CS major is difficult for existing students. CPSC 192 was first offered in Spring 2014. The course is limited to around 20 students. SU is on a ten-week quarter system, and for this course students attended three 75-minute "lectures" (held in a lab) each week.

The 18 students taking the course in Spring 2014 were a diverse group of non-major students, predominantly science and engineering majors, and was a roughly even composition of freshmen through seniors. In the course evaluations, the lab time and direct instructor interaction was well reviewed. About half of the students felt that they might use Python for a future project in their discipline. An in-class informal evaluation showed a large variance in students' view of the "fun-factor" of the assignments, validating the variety of topics included.

4. A DATA PROGRAMMING CS1 COURSE

This section discusses course topics and pedagogy, then describes some of the programming assignments used in the course. While our experience is with teaching an entire CS1 course oriented towards data programming, we expect some instructors may be interested in adopting only parts of our approach or individual assignments.

4.1 Effect on Course Topics and Pedagogy

Supporting students' experience with realistic datasets requires a few changes to the topics, order, and presentation of traditional CS1 material. Students learn similar concepts to those in any other CS1 course, but a slightly different toolset. Our choice of topics is also motivated by viewing our course not just as CS1 but also as CS-Omega: it should give a solid foundation for subsequent practical and theoretical work, but should also be useful even if the student never takes another CS course.

The overall focus in our course is on providing students with the tools necessary to process data provided as files. This focus on processing data provided in files guides the choice and ordering of topics. For example, we introduce the foreach loop but not the for loop with an explicit index. We use file I/O extensively, with only limited use of console I/O. We do not create GUIs nor any other user interface. Students use complex data structures such as dictionaries and graphs, but they do not re-implement basic data structures like linked lists nor algorithms like sorting. We introduce recursion at the end of the term, as an enrichment topic, but it is not used in assignments. We introduce a few concepts that are often missing from CS1 courses but are desirable for data analysis, such as basic statistics and how to plot a graph.

Most assignments gave students some supporting code or libraries. Early in the course, students are given examples of documented, modular code that they may explore at their own pace but are not required to understand until later assignments. We have found it useful to provide a first assignment consisting of a few short programming exercises that does not do any real-world data analysis while students are becoming familiar with an IDE or command line editor and the basics of Python syntax.

We use the Python language, because it is easy to use and is widely adopted in the sciences. Because Python has a significant and usable procedural subset, we do not discuss object orientation. Python's rich set of libraries provides the opportunity to give students valuable experience reading the documentation for a real library and successfully using it. We have used the Python networkX graph library, matplotlib, numpy, scipy, and urllib. However, the overall approach is not tied to Python, and similar assignments could be done with Java, MATLAB, Mathematica, or other programming languages.

At UW, readings were drawn from *Introduction to Computation and Programming Using Python* by Guttag [13], as well as free online resources like *Think Python* by Downey [9] and the Python Tutorial [19]. We also prepared documents on topics such as Python evaluation rules, using the command line, interacting with files, using csv.DictReader, and debugging. At SU, *Python Programming in Context* by Miller and Ranum [17] was the recommended text.

4.2 Sample Programming Assignments

Below we describe several of the assignments used in the courses at our institutions. We list them in the approximate order they were given to students and indicate which assignments were used at which schools. In addition to these assignments, most schools also used a first assignment consisting of a few short exercises that did not do any data analysis. Other assignments used included Twitter sentiment analysis, image analysis, processing stock market data, and linguistic analysis [30].

DNA Analysis (UW, PUC, Evergreen, SU)

DNA can be described as a string over the alphabet of base pairs: {A, C, G, T}. The GC content (the fraction of nucleotides that are either G's or C's) is biologically important. For example, GC content can identify types of genes and can be used in determining classification of species.

In the DNA Analysis assignment, students are given files from a DNA sequencer and are asked to use loops and if statements to count nucleotides, categorize organisms, and compute other statistics about the files they are given. This is students' first look at reading files, although the code for reading files is given to them for this assignment. This is also students' first look at the notion of data cleaning (some DNA sequences contain "junk" base pairs) — an important idea for students to be exposed to in preparation for examining their own data files. Students and instructors have commented that this assignment is a bit simpler than the following ones. While this allows students to succeed early in the course, it could also lull them into thinking future assignments will be as simple.

Oceanographic Data Integration (UW, PUC)

In the Oceanographic Data Integration assignment, students work with real measurements of physical and biological variables from the Puget Sound. The research question they are addressing is "Which environmental variables correlate with the abundance of Ammonia-Oxidizing Archaea (AOA)?" The dataset students are using comes from an NSF-funded project, "Significance of nitrification in shaping planktonic biodiversity in the ocean".

Students are faced with the common task of having to integrate data spread across several CSV files into one file. The provided Python code reads CSV files and computes Pearson correlations, but is missing the bodies of functions to compute mean and standard deviation functions needed by the Pearson code. This assignment introduces students to the common situation where the format of collected data is not always convenient for analysis, and the idea that by reading the data into a program it can be transformed to other formats. Students gain practice with Python lists, functions, loops, and file I/O.

Social Networking (UW, Evergreen)

We are not the first to notice that students are motivated by projects related to social networking [22]. In this assignment, students address the research question, "Which of two collaborative-filtering approaches is better for recommending friends?" The dataset is the Facebook New Orleans social network.

Students are introduced to the idea of a graph as a data structure and use the networkX library both to create a simple graph by hand and to read data from a file into a data structure. We provide scaffolding in the form of function signatures that students need to fill in, as well as a series of assert statements that show the expected output of the functions. Students make use of previous knowledge about file I/O, functions, loops, and conditionals. Learning objectives include using graphs to solve a problem, as well as gaining experience using sets, dictionaries, and sorting. Discussion of various sorting approaches is optional; students only need to know how to use Python's built-in sorting methods.

Election Prediction (UW, Evergreen, SU)

The 2012 US presidential election was a watershed in the fight between pundits and statisticians to accurately predict the outcome of political campaigns. The rivalry became front-page news, with many pundits loudly proclaiming that the statisticians would be humiliated on November 6. In fact, the opposite happened: statistician Nate Silver (of the website FiveThirtyEight.com) correctly predicted the outcome in every state, whereas pundits' predictions varied significantly. In this assignment, students replicate Nate Silver's results by using polling data to predict the outcome of the 2012 US presidential election. Students solve a complex problem using lists, sets and dictionaries and are thus able to appreciate the benefits of good problem decomposition, data structure choice, and testing practices. Most students found this assignment to be quite challenging due to the use of nested data structures. At UW we found providing a preliminary quiz forcing students to examine the data structures closely helped significantly in this respect.

Fraud Detection (UW, PUC, SU)
In the fraud detection assignment, students look for fraud in datasets using two different approaches, broken into two separate assignments. First, students examine the least significant digits of the vote totals in election returns from the disputed 2009 Iranian presidential election. We would expect the digits in the ones and tens place to be uniformly distributed in a valid dataset but students examine the Iranian results themselves and explore issues such as the impact of sample size. In addition to the Iranian data, other election results (e.g. 2008 U.S. presidential election, 2012 Egyptian presidential election) can also be brought in for comparison. In the second part of the assignment, students use Benford's law to evaluate the validity of two datasets: 1) US Census data showing the population of US cities and 2) populations of fictional places from literature and pop culture.

In both parts of the assignment students are guided through an approach to statistics and hypothesis testing through simulation. The basic question posed in this assignment and that students are likely to wrestle with in their own work is "I have observed something. Was it unusual? How unusual?" Our approach to answering this question is to have students write code to generate many possible datasets, then measure the thing of interest in those datasets to produce p-values. Students also plot their results using matplotlib.pyplot. For this assignment students are not given any supporting code thus are asked to write a Python program in good style without a provided template. They make use of loops and lists, as well as string and numerical manipulation.

Estimating Avogadro's Number (SU)
Estimating Avogadro's number using microscopy data and the concept of Brownian motion was a student favorite at SU. In this assignment students were given a sequence of images of polystyrene beads in water and are asked to track the beads to estimate self-diffusion. This was adapted from a Nifty assignment [28] to act as a final project with exposure to post CS1 topics. The objective was threefold: to give students the enthusiasm to try their own future projects, to see some cool science, and to think about data structures. Students were required to conform to complex third party data structures (both B/W & color images), and to create their own data structures (tracking blobs/beads). The introduction of data structures that are open-ended, e.g., "What data do you need to track a bead?", encourages students to stretch their minds around design and data management. As a bonus, a discussion of algorithmic efficiency from too many nested loops can be addressed with more advanced students. Students enjoyed seeing the science in action and the freedom to create and manipulate data in their own way.

Open-ended Final Project (UW, PUC)
At UW and PUC, the final assignment was a multi-week open-ended project addressing a research question of each student's choice. Students could work together in groups of two students. Completion of this project demonstrated a key goal of the course — enabling students to process data to solve real problems of interest to them. At the beginning of the course, we told students about the project and showed example topics addressed by students in previous iterations of the course. This served to motivate students to acquire the tools necessary to accomplish this task throughout the quarter, and it got them thinking about project ideas from day one. Topics addressed by students included the decay mechanisms of the Higgs Boson, the correlation between firearm ownership and violent crimes, school district performance vs. financing, the home court advantage in sports, music classification via note analysis, and forecasting company health from financial statements.

One of the biggest hurdles to picking a research question is finding an appropriate dataset. We required students to use publicly available datasets to enable the course staff to evaluate their work. We provided students with links to quite a few possible datasets to get them started.

We have found it useful to break the project into multiple phases to prevent students from waiting until the last minute and to allow for re-direction of project ideas that are too simple or too ambitious. The first checkpoint requires students to locate a dataset and propose a research question. Later checkpoints ask them to provide background and motivation, flesh out their technique and evaluation method, and finally to present and discuss their results. We have used the final exam period in the course for project presentations. This session became a celebration of the students' accomplishments. Allowing students to see what their peers have accomplished in this manner serves to support the notion that what they have learned in the course can be applied to answer questions from a wide variety of domains.

5. ADAPTING TO YOUR CONTEXT

The first version of the course was piloted at UW in Summer 2012. Since then it has been offered at multiple other institutions. Assignments and course structure were adapted by each instructor to their unique context. Below we discuss issues of interest to instructors wishing to adopt our approach at their institution.

5.1 CS1 with Minimal Infrastructure

At most schools CS1 is one of the largest CS courses offered. Students new to programming are often aided by armies of TAs who have previously taken the course themselves. There may be a dedicated lab that is manned with course staff many hours of the day or there may be hands-on lab sessions scheduled for credit hours. When offering a new or alternative CS1 course, your first few offerings may need to survive without these amenities.

At SU, the course format of "lab in every lecture" offered supervised coding practice in every period. In a 75-minute period, lecture was limited to 45 minutes, and labs were assigned typically at the end of lecture. Overall there were approximately 20 coding exercises used, graded on an effort-only basis. This approach addressed several issues. Given the diversity of backgrounds of these non-major students in terms of major, class standing, and coding experience, it was imperative to require extra coding practice. Additionally, as the tutors at SU know mostly C++ or C#, they were not ideal for helping with the course. This in-class lab time allowed the instructor to fill the tutoring gap for students. Finally, since many students at SU work, they are not able to attend tutoring or office hours regularly, so for some this was the only time to get in-person assistance. Most students appreciated having this time, as it helped reinforce the lecture topics. This mandated extra practice also improved students' coding confidence. This seemed to be especially true for some female students that were initially hesitant to get started for fear of writing "wrong" code on their first attempt. Ultimately, holding lecture in a lab room every period allowed for the flexibility to adapt the time spent on labs to the needs of an individual group of students, and the extra mandated coding was a boon particularly to the weakest students.

Other strategies used at UW and Evergreen to support students new to programming without requiring significant infrastructure included pointing students to online Python resources such as Codecademy.com and CodingBat.com to provide more hands-on practice and immediate feedback. The courses at PUC and UW also used pythontutor.com to help students visualize the structure of data and the effect of control constructs. Overall this is an area to pay attention to as students are likely to be aware of the resources provided to students in your "main" CS1 course and may feel they are being asked to get by with less support.

5.2 Adjusting Assignments to Your Students

As described in Section 4.2, the authors have successfully adapted many of the original assignments from UW for use at their institutions. Here we discuss some of the adjustments made.

In all offerings of the course, students were provided considerable starter code at the beginning of the term, then less and less starter code as the term progressed. (No code was provided for the fraud detection assignment, nor for the final project.) The goal was to make students read well-structured and well-documented code, and to reduce their workload.

However, some students were frustrated when the provided code did not make intuitive sense to them. These students found the assignments' provision of a problem approach and starter code restrictive rather than helpful. At SU, we improved their completion rate by increasing the flexibility of assignments: providing less code, shortening the assignment write-ups, and, in the later part of the course, allowing students to add/remove/rename functions and function parameters.

This introduced students more rapidly to the experiences of problem analysis and program organization, including choosing how to store and manipulate data. It also forced students to better comprehend the assignment goals, specifically to begin by addressing the structure of the data as pre and post conditions. This flexibility made the starter code smaller and more approachable. On the other hand, these sparser specifications required more in-class time be spent on explanations, they reduced the utility of staff test suites, and they required more effort to grade.

In all offerings, we found the "wishful thinking" approach to problem decomposition effective: when faced with an unsolved part of the problem, the students could name and specify a routine that would solve it, use that hypothetical routine, and come back later to implement it [1]. A live coding lecture demonstrating "wishful thinking" was effective in encouraging students to write additional functions. We grounded this method in practices that are common to students, such as outlining a paper before writing.

At Evergreen, the difficulty of the material was challenging to students, some of whom worked multiple jobs or had to support families. These students could not devote their full attention to learning as much as younger students at traditional universities. To adapt the original data programming course, we only selected four of the assignments and subdivided each of those in half to create eight mini-assignments. Turn-in dates were flexible, and students were allowed to time-box their efforts (for example, 11 hours of outside time per week) to attempt as many problems as possible within that time. The extent to which students were able to complete assignments provided valuable data to adjust the difficulty of assignments for this demographic in the future.

None of our offerings have used all of the assignments we have collected. Thus, expanding the course from a quarter to a semester can be easily accomplished by adding assignments, expanding the final project, or reducing the pace. One could also add enrichment topics, such as an introduction to object orientation — so long as the focus is on how it is useful for real-world data analysis.

5.3 Non-majors and Non-traditional Students

The course delivery at SU was specifically focused on catering to non-majors, which presented the additional challenge of motivating students who view the course as only secondarily important. The style of this course, which focuses on practicality for students taking only one CS course, was helpful in that respect. However, many students were surprised by how much time it took to master data programming. Interesting assignments were key to motivating students outside of class. Interactive in-class activities were the solution of choice to keep students alert and attentive during lecture. Methods that were successful and minimally interruptive at SU included code something very small and report before continuing lecture (using the interpreter), have everyone stand and then vote by sitting, using a suit of cards to define a sorting algorithm, and writing algorithms on paper.

At Evergreen, the data programming approach was especially appealing to both non-majors (who were initially interested in applying CS to other sciences such as biochemistry) and non-traditional students. Older adult students appreciated the real-life applications, which seemed less contrived or condescending than "hello world" examples or toy problems. Subdividing assignments into smaller pieces and keeping due dates flexible were important for keeping this group of students motivated and engaged.

5.4 Articulation into (a non-Python) CS2

While designed to serve students well as both the first and last programming course they may take, we have found the course has also worked well for students pursuing more CS courses, usually offered in a language other than Python. Several institutions use Python in their CS1 courses and some have reported on their experience with students articulating into a non-Python CS2 [10].

At UW, where the course has now been taught three times, we have seen a number of students from the class go on to major in CS (and serve as TAs for the course). During the first offering of the course, we held a parallel one-credit Java basics course with the goal of helping prepare students to take CS2 in Java. At UW we also offer a one-quarter combined CS1/CS2 course in Java that is an ideal follow-on course for students interested in majoring in CS after taking our course, since they already have most of the conceptual background provided by the CS1 course but are not familiar with the Java terminology and toolset.

At Evergreen, where the data programming course served as the first course for both CS majors and non-majors alike, the change from Python in the fall quarter to Java in the winter quarter required several adjustments. One difference is that our data programming assignments provided more supporting code than is typical for a CS1 or CS2 course. When moving to the next course, Evergreen students had to adjust to implement more from scratch.

At SU, CS majors take CS1 and CS2 courses offered in C++ (the primary language used in the major). SU plans to offer an alternative CS2 to follow the data programming course which will include an introduction to C++. The goal is to serve CS minors and non-majors who have to take additional CS courses. This will mean that people wanting to switch to the CS major after taking data programming can do so without having "wasted" a course. This alternate path is garnering significant interest from previous and prospective students of the data programming course.

At PUC, the next course students take after the data programming course is a "breadth-first CS1" [4] that assumes prior programming experience. This second course expects that students do not come in knowing the language required for the course and includes a transition to another language, usually C.

6. CONCLUSION

Students in CS1 can do real-world data analysis. We have described how multiple diverse schools have successfully implemented the data programming approach. We welcome other instructors to join in the fun! Course syllabi, assignments, and other resources can be found at our instructor resources site [30].

7. REFERENCES

[1] H. Abelson, G. J. Sussman, J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1996.

[2] J.C. Adams and R.J. Pruim. Computing for STEM majors: enhancing non CS majors' computing skills. In *SIGCSE 2012*, 457-462.

[3] R. Catrambone and M. Guzdial. Computational freakonomics. http://swiki.cc.gatech.edu/compfreak, 2012.

[4] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. Joe Turner, P. R. Young. Computing as a discipline. *Commun. ACM 32*, 1 (January 1989), 9-23.

[5] S. Cooper and S. Cunningham. Teaching computer science in context. *ACM Inroads* 1, 1 (March 2010), 5-8.

[6] W.P. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice* (2 ed.). Prentice Hall Press, 2008.

[7] P. DePasquale. Exploiting on-line data sources as the basis of programming projects. In *SIGCSE 2006*, 283-287.

[8] Z. Dodds, R. Libeskind-Hadas, C. Alvarado, and G. Kuenning. Evaluating a breadth-first cs 1 for scientists. In *SIGCSE 2008*, 266-270.

[9] A. B. Downey. Think Python, http://www.greenteapress.com/thinkpython/, 2012.

[10] R.J. Enbody, W.F. Punch, and M. McCullen. Python CS1 as preparation for C++ CS2. In *SIGCSE 2009*, 116-120.

[11] A. Erkan, T. Pfaff, J. Hamilton, and M. Rogers. Sustainability themed problem solving in data structures and algorithms. In *SIGCSE 2012*, 9-14.

[12] M. Goldweber, J. Barr, T. Clear, R. Davoli, S. Mann, E. Patitsas, and S. Portnoff. A framework for enhancing the social good in computing education: a values approach. *ACM Inroads* 4, 1 (March 2013), 58-79.

[13] J. V. Guttag. *Introduction to Computation and Programming Using Python*. MIT Press, 2013.

[14] S. Hambrusch, C. Hoffmann, J.T. Korb, M. Haugan, and A.L. Hosking. A multidisciplinary approach towards computational thinking for science majors. *SIGCSE 2009*, 183-187.

[15] P.B. Lawhead, M.E. Duncan, C.G. Bland, M. Goldweber, M. Schep, D.J. Barnes, and R.G. Hollingsworth. A road map for teaching introductory programming using LEGO© Mindstorms robots. *SIGCSE Bull.* 35, 2(June 2002),191-201.

[16] S. Lohr. The age of big data. *New York Times*, Feb 2012.

[17] B. N. Miller and D. L. Ranum. *Python Programming In Context*. Jones & Bartlett, 2013.

[18] Princeton Univeristy, Computer Science 126: General Computer Science, http://www.cs.princeton.edu/~cos126

[19] The Python Tutorial. https://docs.python.org/2/tutorial/

[20] L. Rich, H. Perry, and M. Guzdial. A CS1 course designed to address interests of women. In *SIGCSE 2004*, 190-194.

[21] K. A. Robbins, D. M. Senseman, and P. E. Pate. Teaching biologists to compute using data visualization. In *SIGCSE 2011*, 335-340.

[22] M. Sahami, FacePamphlet: Implementing a Simple Social Network, Retrieved Dec 1, 2014 http://nifty.stanford.edu/2009/sahami-face-pamphlet/

[23] D. E. Stevenson and P. J. Wagner. Developing real-world programming assignments for CS1. In *ITICSE '06*, 158-162.

[24] D. E. Stevenson, M. R. Wick, and S. J. Ratering. Steganography and cartography: interesting assignments that reinforce machine representation, bit manipulation, and discrete structures concepts. In *SIGCSE 2005*, 277-281.

[25] D. G. Sullivan. A data-centric introduction to computer science for non-majors. In *SIGCSE 2013*, 71-76.

[26] S. L. Tanimoto. *An Interdisciplinary Introduction to Image Processing: Pixels, Numbers, and Programs.* MIT Press, 2012.

[27] Teaching Lab Skills for Scientific Computing, Retrieved Dec. 1, 2014 http://software-carpentry.org/

[28] K. Wayne, Estimating Avogadro's Number, Retrieved Dec. 1, 2014 http://nifty.stanford.edu/2013/wayne-avogadro.html

[29] G. Wilson, C. Alvarado, J. Campbell, R. Landau, and R. Sedgewick. CS-1 for scientists. In *SIGCSE 2008*, 36-37.

[30] Data Programming Instructor Resources, http://tinyurl.com/dataprogramming/instructor-resources