Data-Centricity: A Challenge and Opportunity for Computing Education

Shriram Krishnamurthi and Kathi Fisler

Rethinking the content of introductory computing around a data-centric approach to better engage and support a diversity of students.

On a growing number of campuses, data science programs offer introductory courses that include a non-trivial amount of programming. The content of such courses overlaps that of traditional computer science introductory courses, but neither course subsumes the other. This situation creates challenges for students. Introductory courses should help students decide which disciplines to pursue further, but misaligned data science and computer science introductions leave students unable to switch between areas without starting over. This in turn puts pressure on departments to figure out how to accommodate students as they explore their curricular options. In universities where finances follow student enrollments, overlaps such as these can also lead to turf wars and other tensions that adversely impact students.

We view "data science" as the process of answering questions about the world through the application of (usually statistical) computational methods to data. Data scientists benefit from some computing background. In recent years we've also seen the rise of the related profession of "data engineers", who need a substantial computing background. In addition, the central role of data across computing demands CS majors with basic data-science skills. Therefore, for the sake of this broad spectrum of students, it is time to rethink the content of introductory computing. We believe the approach described in this article, *data-centric introductory computing*, can support and engage students with diverse interests.

Introductory Data Science as a Critique of Introductory Computing

A conventional CS1 [1] which emphasizes control structures and imperative programming aligns poorly with the learning goals of aspiring data engineers or data scientists. Data science uses rich functions that transform and compute with sophisticated data, both of which are far from the focus of CS1. The explicit manipulation of aggregated data—such as cleaning, splitting, or refactoring—is best done through operations that more closely resemble queries and higher-order functions [4], both of which occur much later in traditional curricula. Data science works with *real* datasets that touch on real-world problems, rather than artificial starter problems based on numbers, strings, and arrays. In all of these ways, conventional CS1 courses appear inauthentic or irrelevant for data-facing students. Moreover, the end goal of conventional early CS courses is usually to prepare students for more CS courses, rather than to prepare them to continue to learn additional data-relevant programming content as their interests demand.

However, CS1 courses that eschew these trends don't only fail to engage the budding data scientist: they also do a disservice to computer science students. There are innumerable data sets that cover a very broad span of human

interests, from politics and economics to sports and voting in music competitions. Through their use, computing can be seen even by novices for what it now is: a field that engages with humans and the world. This alters some of the standard perceptions that keep away many groups of students. The introduction of data also makes it easy to concretely illustrate the social consequences and perils of data-driven decision making, even to novices. Thus, *we should view the rise of data science curricula as a criticism of computing curricula, and work to address our shortcomings*.

Why Not a Separate Data Science Track?

Given these weaknesses of CS1 (which often extend into CS2), perhaps the rise of parallel data science tracks (let's call them DST) presents the solution? Sadly, the DST courses we have reviewed take a rather ad hoc view of computing and programming. There is little emphasis on program design, software testing, or the impact of data structure on complexity. These are not merely important issues: they are critical. Even small programs may end up influencing policy decisions or research findings. We have to be able to trust the code and the results.

Many data scientists, and especially data engineers, will want to take more (traditional) CS courses such as cloud computing, security and privacy, software engineering, visualization, and databases. The limited computing preparation that a DST would offer leads to one of three outcomes, all unhappy:

1. Each of these courses needs to be reconstructed in a separate "data science department" or equivalent, resulting in a huge duplication of labor and cost (which is simply infeasible at many institutions).
2. Students will arrive in these upper-level computing courses without the relevant computing prerequisites. These students will have vastly poorer educational outcomes and create a significant burden for faculty.
3. Those upper-level courses will have to scale back their prerequisites, resulting in weaker curricula for traditional computing students.

In general, many students who are offered both DST and CS1 options lack sufficient understanding of either discipline to predict which focus best aligns with their interests. In educational systems (like in the US) where students can easily move between disciplines in college, the choice between DST and CS1-CS2 is premature.

Reform Introductory Computing!

We suggest that there is a strong alternative: to integrate data-science components into introductory computing. This requires a significant re-thinking of what introductory computing looks like, and is a major departure from the traditional CS1-CS2 structure. What might this rethinking entail?

We should begin the curriculum *in "data science"* with some basic data engineering. That is, students begin right away with data sets of some complexity reflecting real-world questions. Perhaps surprisingly, we believe that even the choice of representation matters, and recommend focusing specifically on data represented in *tabular* formats. This offers many advantages:

1. Innumerable real-world data are published as tables. Students can pick data sets of interest to them (within some constraints), enabling them to personalize their education and feel more invested in it. Exercises become much less artificial.
2. Even quite young students understand tables instinctively [2], and have experienced them in everything from middle-school math classes to spreadsheets.
3. Despite this, tables are a fairly sophisticated data structure: an unbounded homogenous sequence of bounded heterogeneous structures. Getting to this point in a traditional computing curriculum takes quite a while.
4. Tables (especially tidy [5] ones) are inherently parsed, eliminating a complex and imprecise step that greatly complicates other interesting data formats like text.

In short, tables are a "sweet spot" in introductory computing.

Doing useful work also requires some basic statistics. However, most students enter college with a rudimentary knowledge of some operations such as central tendencies. In our experience, even mathematically-nervous non-majors are able to successfully apply them with reasonable amounts of support. Teaching basic visualization generates meaningful artifacts beyond code. Furthermore, showing students core programming concepts such as conditionals (for selecting data or identifying data-entry errors) and functions (for running experiments on datasets with similar column structure) not only enables them to complete projects with experimental data analysis, it also grounds typical programming content in a rich and meaningful context.

From Data to the Rest of Computing

Tables are, of course, not a universally appropriate data structure. Once students start engaging with the world, they soon encounter many other kinds of natural structures to represent: whether sport competition brackets or genealogical information or travel networks (respectively, usually, trees, DAGs, and graphs). While these can be encoded in tables, this requires varying degrees of inelegance, which students can easily see.

This, of course, is where computing really shines. For decades we have built theory and languages for complex data structures: to represent them, program over them, and reason about them. Thus, these data structures—which represent a limitation of tables—are a perfect point of transition to traditional computer science, through data structures and systematic programs over them. Not only are non-tabular datatypes more natural in these cases, they also offer different affordances, and can even confer significant performance advantages. We can get to these points potentially earlier than we would in traditional CS1-CS2; with much greater motivation; and with students already well-equipped to do useful work through their exposure to tables. From a computing perspective, this should be regarded as a win-win.

While data scientists arguably don't need all these data structures, we believe exposing students to the limitations of tables through concrete data-facing examples is valuable in and of itself. This also represents a branch point in the curriculum. Those certain they do *not* want to learn more computing can move on to statistics and other disciplines (armed with a solid introduction to programming and computing), while the rest can proceed with a more traditional computing path.

Data-Centricity

In short, we propose a restructuring of traditional CS1-CS2 with rudimentary data science preceding, and fluidly leading to, more traditional data structures. We call this approach *data-centric computing*, with the formula

Data-centric computing = data science + data structures

(in that order: this + is not commutative). We have been prototyping this approach for a few years now at Brown, with great success, and have learned a good deal from it.

This shift certainly presents challenges. It requires a fairly different introductory pedagogy that has not been developed well in the computing education community. Everything from programming methodologies to problem decomposition to topic ordering, and the interleaving of statistics with programming, is open and needs study. Mathematics education techniques like notice-and-wonder [3], which was popularized for data reasoning by the *New York Times* [6], become relevant. Students need to be taught to contend with precision and accuracy; we have to determine the right way to introduce data cleansing; and so on. We therefore call on the computing education community to rethink its approaches, prescriptions, and open questions.

The Promise

Moving introductory computing to data-centricity can open many doors. Other subjects, given an accessible, useful introductory computing course, may rethink their approach. A new generation of students will be able to communicate across disciplines in the language of computation. Computing's application to a wide variety of disciplines will stand up front and center, without compromising the core of our subject. Our goal, therefore, should be to create *at scale* the new kinds of partnerships that currently only happen in very specialized settings.

A data-centric introduction to computing also buys students time in curricula that give them choice. They don't have to prematurely commit to "data science" or "computing", which they are likely doing on the basis of what they've heard from parents, friends, and newspaper articles. Instead, they can experience the two subjects together for themselves before having to choose between the two (and, even if they pick a third, they will have a useful, applied understanding of computing).

Even in a time of exploding enrollments, we owe it to our students to ask these questions and act on the answers. It's easy to think we are successful in a hot marketplace, but the integration of data science could create curricula of lasting value even in a weak one. At some (perhaps more resource-rich) institutions this will take the path of whole-cloth reform; at others (more resource-starved), educators will have to find an incremental path. All this provides an opportunity and imperative for educators and researchers, and (reflecting the breadth of the field) contributions can come from people with many different kinds of expertise (pedagogy, statistics, visualization, programming languages, and more).

References

1. The Joint Task Force on Computing Curricula Association for Computing Machinery and the IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.*
2. Konold, C., Finzer, W. and Kreetong, K. *Modeling as a core component of structuring data*. Statistics Education Research Journal 16, 2 (Nov. 2017), 191–212.
3. Ray-Riek, M. *Powerful Problem Solving: Activities for Sense Making with the Mathematical Practices*. Heineman, 2013.
4. Wickham, H. *Advanced R. Second edition*. Chapman and Hall/CRC The R Series, 2019.
5. Wickham, H. *Tidy Data*. Journal of Statistical Software 54, 10 (2014).
6. https://www.nytimes.com/2017/09/06/learning/announcing-a-new-monthly-feature-whats-going-on-in-this-graph.html