Motivating Introductory Computing with Student-Driven Datasets

Austin Cory Bart

June 30, 2015

Contents

List of Figures

1 Summary

This preliminary proposal explores introductory computing contexts, and suggests a research plan to evaluate its motivational impact on students. In particular, it focuses on the use of Datasets and Data Science as an introductory context as a holistically motivating context that can authentically appeal to a wide ranger of learners. I propose to design pedagogical materials and technologies to integrate datasets into courses. This research will move past traditional research on using data science for education, which currently is limited to brief case studies with little to no evaluation. Instead, it will focus on serious evaluation and the principled, detailed analysis of how the affordances, issues, and value of this context. It will be comparative to existing research on other introductory contexts such as Media Computation or Game/Animation Design. Additionally, it will have a strong focus on new technological solutions to support the learning process, leveraging computational techniques in software engineering and data analysis. It will also contribute to the ongoing research onto the effect of motivation on academic success, in particular the novel application of the MUSIC Model.

2 Problem Statement

Computing is increasingly considered a 21st century competency, paralleling its growing entrenchment within universities general education curriculum [62]. At Virginia Tech, for instance, the core requirements at the university have recently shifted to require all undergraduate students to take credit hours in "Computational Thinking" – emphasizing crucial topics in Abstraction, Algorithms, Computing Ethics, and more. The students in this course represent a diverse spread of majors from the arts, the sciences, engineering, agriculture, and more.

Although they have rich backgrounds, non-major learners are a challenge for instructors because they have no prior background in computer science, are not confident about their ability to succeed in the course, and have no assurances that it will be a useful experience. These students are at-risk for dangerously low levels of motivation. As a more mature audience with domain-identified career goals and needs, existing interest-driven approaches to introductory computing may not be sufficiently effective [28]. How do we motivate these unique learners?

This work will develop new pedagogical approaches and technical tools to better understand and resolve this question. In particular, I will explore the efficacy of Student-driven Data Science as an introductory context. This context moves beyond basic models of motivation to take advantage of multiple dimensions of engagement. Although this idea has been known for a long time as successful for advanced computing courses, I propose new technology and pedagogy to enable its use in the introductory level, where it is most effective. Further, I will conduct more comprehensive and fruitful evaluations than exists in the current literature.

3 Theories of Motivation

There are many theories of how people learn, how to teach, and how to engage with students. In this proposal, I investigate Academic Motivation - the process by which people choose to direct energy towards learning. Motivation is well understood to be a crucial part of education, particularly in Computer Science [11].

3.1 MUSIC Model of Academic Motivation

In this preliminary proposal, I lean on the MUSIC Model of Academic Motivation as a primary framework. The decision to use the MUSIC model was based on several criteria. Although there are many motivational models available, few strive to be holistic models specifically developed for academics. For example, theories like Expectancy-Value and Cognitive Evaluation Theory have a wider scope and have stemmed from other disciplines such as healthcare. The MUSIC Model is derived from a meta-analysis of these other theories, incorporating only the academically relevant components. Further, the MUSIC model is a tool meant for both design and evaluation, allowing it to be used in all phases of this work. Finally, the model and its associated instrument, the MUSIC Model of Academic Motivation Inventory (MMAMI) , has been extensively validated and utilized in other educational domains, making it a reliable device [36].

The MUSIC model identifies five key constructs in motivating students [35]:

eMpowerment: The amount of control that a student feels that they have over their learning – e.g., course assignments, lecture topics, etc. This is amount of Agency that a student

percieves.

Usefulness: The expectation of the student that the material they are learning will be valuable to their short- and long- term goals. There is no clear delineation of the time-scale for these goals, but there is nonetheless a distinction between strategic skills that students need to be successful in careers and personal interests and the tactical skills they need to complete present-day tasks.

Success: The student's belief in their own ability to complete assignments, projects, and other elements of a course with the investment of a reasonable, fulfilling amount of work. Most students desire experiences that are successful, but still challenging enough to be worth it.

Interest: The student's perception of how the assignment appeals to situational or long-term, individual interests. The former covers the aspects of a course related to attention, while the latter covers topics related to the fully-identified areas of focuses of the student. It can be difficult to parse out the difference between individual interests and long-term career goals – there can be alignment between these two components.

Caring: The students perception of other stakeholders' attitudes toward them. These stakeholders primarily include their instructor and classmates, but also can be extended to consider other members of their learning experience (e.g., administration, external experts, etc.). It can also be viewed as the extension towards society as a whole.

Students are motivated when one or more of these constructs is sufficiently activated. They are not all required to achieve maximal levels, and in fact that is not always desired – it is possible, for instance, for a student to feel too empowered, and become overwhelmed by possibilities. For some of these constructs, a careful balance is required, and it may not be possible to ever achieve a minimal level; no matter how exciting you make your lecture, you may never convince your students it is interesting, although it is possible that they will still consider it useful and stay motivated. Crucially, students' subjective *perception* of these constructs is a defining requirement and is more important than objective reality. It does not matter if you believe that your lecture is Useful, if you have not convinced your students that it is (although an instructor's beliefs are a powerful tool for convincing their students).

The MUSIC model is often used as an organizational framework and an evaluative tool. As the former, it is a list of factors to consider when building modules, assignments, and content of a course. At all times, instructors can consider whether they are leveraging at least one construct to motivate their students. As the latter, it offers both a quantified instrument (MMAMI) and a structure to anchor a qualitative investigation on. The model has also directly been used tactically in course design: Jones describes a controlled classroom experiment to motivate students by having an experimental group reflect on how a course satisfies the constructs of the MUSIC model. Students were simply prompted to answer the question "How will the material presented here will be useful to you?". Quantitative data gathered after the experiment indicated a significant increase in motivation [42]. The MUSIC model is therefore a very flexible resource for studying motivation.

The MUSIC model has not been seriously applied to Computer Science education before, although a number of other motivational frameworks have been leveraged in studies. Many of these

studies often find results that hint at many of the elements of the MUSIC model. For instance, Mitchell [43] interest, usefulness, and "intellectual challenges" (success) as three primary elements influencing success in computer science. Although they do not discuss empowerment and social elements, many other studies often suggest these themes. Still, it should be acknowledged that it is an open question whether the MUSIC model can be applied directly, or if this domain has different characteristics from other disciplines.

3.2 Situated Learning Theory and Authenticity

Although most of this proposal relies on the MUSIC Model of Academic Motivation, I will discuss Situated Learning Theory briefly. Situated Learning Theory, originally proposed by Lave and Wenger, argues that learning normally occurs as a function of the activity, context, and culture in which it is situated [39]. Therefore, tasks in the learning environment should parallel real-world tasks, in order to maximize the *authenticity*. Contextualization is key in these settings, as opposed to decontextualized (or "inert") settings. The key difference is that learning is driven by the problem being solved, rather than the tools available therefore, the problem being solved should lead directly to the tool being taught. This argument suggests that this is a key aspect of motivating students.

Authenticity is a crucial, recurring theme within Situated Learning Theory. All instruction and assessment must be aligned with reality such that success in the former leads to success in the latter. However, there is a subtle nuance here – authenticity is a perceived trait, not an objective one. Students derive value from their learning only if they *perceive* authenticity, regardless of whether the instructor has successfully authenticated the experience.

Situated Learning Theory has been applied to the domain of Computer Science before, with mixed results. A seminal paper by Ben-Ari [7] explores its application and limitations. This paper is somewhat hasty in its application of SL Theory by taking a macro-level view – they narrowly look to Open-Source and Industry Software Development communities as the only potential CoPs and interpret SL Theory as strictly requiring constant legitimacy, largely ignoring the possibility for gradual development of authenticity within individual courses and modules throughout a curriculum:

> What I am claiming is that situated learning as presented in their work cannot be accepted at face value, because it simply ignores the enormous gap between the world of education and the world of high-tech CoPs, which demand extensive knowledge of both CS subjects and applications areas. This gap can only be bridged by classical decontextualized teaching in high schools, colleges and universities.

However, other researchers have found it a useful lens for analyzing curricula. For instance, Guzdial and Tew [28] used the theory to innovatively explore and deal with the problem of inauthenticity within their Media Computation project. SL Theory clearly has value, but only as a function of the way that it is applied. In this preliminary proposal, I will take advantage of SL Theory as a generalized tool for exploring the topic of authenticity throughout introductory Computer Science.

The original work in Situated Learning Theory was categorically not about pedagogy or instructional design- it described how people learn and the importance of context and collaboration, but

it did not recommend a particular style of classroom. Subsequent research by Brown [8] and others expanded the theory so that it could be applied to the design of learning experiences. These expansions often naturally dictate the use of active learning techniques, reducing the role of lecture in favor of collaborative, problem-based learning activities. Choi & Hannafin [13] describe a particularly useful, concrete framework for designing situated learning environments and experiences. The Choi & Hannafin framework has four key principles:

Context "... The problem's physical and conceptual structure as well as the purpose of activity and the social milieu in which it is embedded" [52], context is driven not just by the atmosphere of the problem at hand, but also by the background and culture surrounding the problem. A good context enables a student to find recognizable elements and build on prior understanding, eventually being able to freely transfer their learning to new contexts.

Content The information intending to be conveyed to the students. If context is the backdrop to the learning, then content might be seen as the plot. Naturally, context and content are deeply intertwined with each other, and its difficult to talk about one without referencing the other; in fact, content is an abstract entity that needs to be made concrete through contextualization when it is delivered to the learner. If the information is too abstract, than it will never connect with the learner and will not be transferable to new domain. However, if it is too grounded in a domain, then it will not be clear how it can be re-applied elsewhere. Ultimately, content must be given in a variety of forms to maximize transfer. Two useful methods for building content are anchored instruction (exploring scenarios, or anchors, in the context based on the content) and cognitive apprenticeship (mediating knowledge from an expert to the novice learner in a mentoring relationship).

Facilitations The modifications to the learning experience that support and accelerate learning. Facilitations provide opportunities for students to internalize what they are learning by lowering the barriers that can surround situated experiences, possibly at the cost of some amount of authenticity. These modifications might be technological in nature, but they can also be pedagogical. Although there are many different forms that Facilitations can take, Scaffolding is one of the most common. Scaffolding is a form of support that is intended to extend what a learner can accomplish on their own. This support is required at the onset of the learning process, but is unnecessary once a sufficient threshold has been passed; during this transition, the amount of scaffolding can be tuned to the learners understanding. In Computer Science, for instance, students often take advantage of software libraries and frameworks to create sophisticated graphical programs that would be beyond daunting if implemented from scratch.

Assessment The methods used to assess the learning experience and the progress of the student. Choi & Hannafin gives special attention to the teach to the test problem, and how assessment needs to change to measure students' ability to solve authentic problem (as opposed to their ability to solve the tests specific problem), and to be able to transfer their understanding when solving different but related problems. It is important that assessment is measured against the individualized goals and progress of a learner, requiring that any standards used be fluid and adaptable to different learners personal situations. Of course, assessment should

be an on-going part of the learning process, providing feedback and diagnostics. Ultimately, the learner should join in the process of assessment as they transition to an expert being able to meta-cognitively self-evaluate the effectiveness of ones methods and communicate results to others are key abilities of experts.

3.3 Contexts vs. Content

There is a reciprocal relationship between contexts and content. Figure 1 demonstrates an example of this relationship for the expected emphasis on big data as a context vs. content throughout an undergraduate curriculum, from a CS-0 (non-majors) course all the way to an upper-level course specifically on big data. Just as the upper-level course would naturally use big data as its context and content, a CS-0 course could still have content related to big data. However, the majority of the use of big data would be as the framing story for assignments, especially in earlier parts of the course. When students learn programming in the context of, say, game development, they are almost necessarily learning content related to game development that may not be universal to computer science – e.g., how graphical resources are organized and accessed within the game engine. This content may be seen as a distraction by the instructor, or as useful side knowledge - for example, if a student had to learn how to use a command line in order to compile their game, they would be learning an authentic skill that might not be considered part of the core content, but is nonetheless generally useful. When evaluating a context, it is useful to consider what content it represents, and how authentic and useful it is. The authenticity of content that is attached to a context affects the authenticity of the learning environment as a whole.

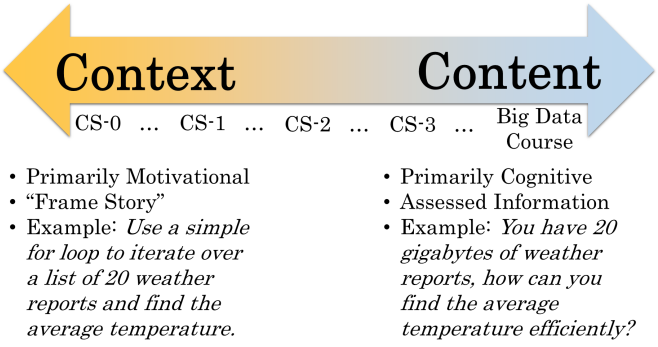


Figure 1. Content vs. Context

Fascinatingly, the need for a strong context diminishes as learners mature and become domain-identified – the content itself becomes the context. Learners start to see other contexts as nothing more than distractions and unnecessary fluff. This makes sense – you would hope that Computer Science majors in their third semester would be naturally interested in the material, and this is borne out in experimental data. For instance, Yarosh & Guzdial attempted to integrate Media Computation in a CS-2 (Data Structures) course, and found that the learners had "outgrown the desire for a context" [63]. These results are similar to results we found in our interventions with a CS-3 (Data Structures and Algorithms), where students seemed more irked by the surrounding context than intrigued.

Of course, it is up to instructor to determine the depth and breadth of the context's integration. The trade-off between the value and distraction added by the context is a delicate formula. Consider the scenario in figure 2. A steel wall is a relatively relatable concept for most students – they can readily imagine such a large, durable object, and it somewhat reasonable to expect that objects would interfere with it. In this scenario, the instructors consider replacing the wall with a comic book character – something that they anticipate will be more "fun". If they are in tune

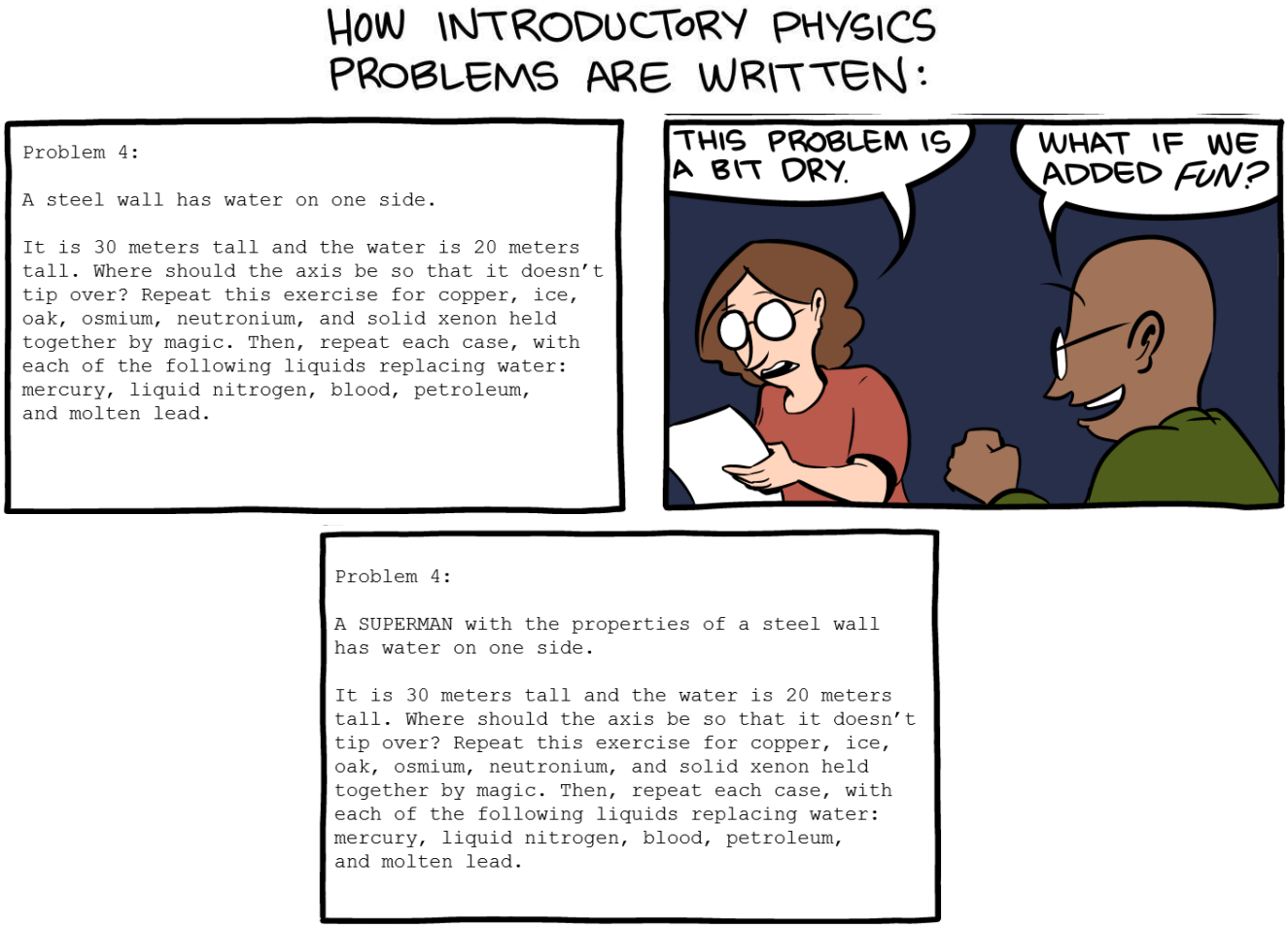


Figure 2. How to Add Fun to Education

Making the context "Fun" is not necessarily trivial, whether in physics education or computer science education. [61]

with their learners, this might actually be an effective context – perhaps they know that their learners are comic book fans. However, because the integration is only at the surface level, it is possible that the learners will see this as a forced reference, and they will have a more negative reaction. It is also possible that they will not recognize the reference, or feel no positive emotions with it – many contexts do not take into account gender, racial, or socio-economic characteristics of the anticipated learner. I suggest that the process of motivating students using a context is non-trivial, and in the following section I will explore prior work in contexts for Computer Science Education.

4 Literature Review

4.1 Introductory Computing Content

There is a serious, on-going debate about what novice students in computing should learn, with limited consensus. Complicating this discussion is the bifurcation of undergraduate introductory computing into Computational Thinking (sometimes referred to as CS-0) and Computer Science (sometimes referred to as CS-1), and the entirely different curriculum proposed for K-12 education. There are some commonly agreed aspects however: any good introductory computing course will cover abstraction (representing complex phenomena more simply, usually as coded data), some form of decision-making (e.g., `if` statements, `cond` statements), and some form of iteration (e.g., `for` loops, recursion) [24,38,47]. From there, different curriculums lay out the material differently. The "How to Design Programs" curriculum emphasizes a functional programming model, using a LISP-descended language named Racket, and the only looping mechanism that students are taught is recursion. The dominance of the Object-Oriented Model in software engineering usually leads to a strong emphasis in introductory courses on abstracting data using Objects and Classes – in fact, there is even an "Objects First" curriculum. Of course, besides the technical aspects, definitions include softer skills such as a tolerance for unstructured problems and collaborative attitudes [24, 33].

In general, this proposal will not take a strong view on what should be included in an introductory experience, beyond a core of Abstraction and Algorithms. However, for practical purposes, materials and research work are grounded and tested in a course, and have adaptions for that content. Specifically, this proposal is used in a Computational Thinking course, so that is how the content is oriented. "Computational Thinking" was coined by Seymour Papert in 1993 [48] and popularized by Dr. Jeannette Wing's 2006 paper [62], which opened a floodgate of discussion about the term. Unfortunately, there is still limited consensus on *what* exactly CT is, whether it should be universally taught, how it should be taught, and how to identify when it has been taught.

An excellent resource for summarizing the history of Computational Thinking research is the 2013 dissertation by Wienberg [60]. This comprehensive survey analyzed 6906 papers directly or indirectly related to Computational Thinking from 2006-2011, describing research efforts and findings. Over half of the research on CT describes approaches to pedagogy (Curriculum and Program Description), leaving a small amount to modeling (Philosophy and Opinion) and assessment (Research and Evaluation). The lack of assessment research is understandable given the youth of this area of research, but still troubling. Even more troubling, however, is the further analysis of the 57 empirical studies. Only fifteen (26%) studies include or sought an operational definition of computational thinking, and only six go beyond the superficial (solely describing computational thinking as a "way of thinking", a "fundamental skill", or a "way of solving problems"). The failure to identify an operational definition weakens the theoretical strength of the studies. This weakness likely stems from the background of the researchers: 28% of the articles involved non-CS majors and **only 18% of the articles involved education experts.** In other words, over four-fifths of this educationally-oriented research was performed by people with no real formal training in educational research techniques. This is particularly troubling given that Computational Thinking is a strong target for interdisciplinary endevaors.

Weinberg reflects on the continuing debate about the importance of Computational Thinking:

> Many, like Wing, believe computational thinking to be a revolutionary concept, one as important to a solid educational foundation as are reading, writing, and arithmetic (Bundy, 2007 [10]) (Day, 2011 [15]). Others believe its potential and significance are overstated (Denning, 2009 [16]; Hemmendinger, 2010 [31]), and some have voiced concern that by joining forces with other disciplines computer science might be diluting either one or both of the participating disciplines (Cassel, 2011 [12]; Jacobs, 2009 [34]). Both the praise and the criticism for computational thinking could perhaps be tempered by reflecting on a historical quote by Pfeiffer in 1962: Computers are too important to overrate or underrate. There is no real point in sensationalizing or exaggerating activities which are striking enough without embellishment. There is no point in belittling either. (Pfeiffer, 1962 [50]).

Although it is ambiguous what Computational Thinking is, we will take it as a given that it requires students to learn some amount of non-trivial programming.

4.2 Introductory Computing Contexts

As part of the overarching goal to bring more students into Computer Science, a large number of contexts have been explored in Introductory computing. The context of a learning experience grounds the learner in what they already known, in order to teach the new material. Many introductory computing experiences focused on presenting the content as purely as possible, which can come across as abstract and detached [64]. However, starting with Seymour Papert's work with robotics and the LOGO programming environment in the 70s [48], instructors have been interested in motivating students' first experience with richer contexts. Some of these contexts rely on Situational Interest (e.g., Digital Media "Computation" (Manipulation) [25] and Game Design [64]), while others attempt to provide enduring career value (e.g., [Big] Data Science [2]) or short-term social applicability (e.g,. Problem Solving for Social Good [26]). Ultimately, each of these approaches draws on different facets of motivation – they do not partition the space, but reside in it organically depending on their implementation. In this section, I will discuss the implications of these different approaches.

4.2.1 Abstract Contexts

Denning describes the early perception of Computer Science by the public as "stodgy and nerdy" [17], since many early computer science classes were driven so strongly by mathematics and logic. A common early introductory programming problem, for instance, is writing a function to compute a fibonacci number – a relatively simple task if you are familiar with the recurrence, and one that leads quite nicely to discussions on the implementation of algorithms, computational complexity, and a host of other subjects [44]. These contexts are "abstract" because they are already at a similar level of abstraction as the content they are attempting to convey. However, Oliveira [46] suggests that the discussion about "abstract vs. concrete" contexts is a misleading one, because the purity with relation to the content is less important than *prior knowledge*. According to modern constructivist and cognitivist theories, learners build on prior knowledge, and the ability

to relate to what they know is crucial. The simple fact is that most students are not particularly good at mathematics, so relying on it as a context is not a useful approach compared to finding subjects that students know and understand readily.

4.2.2 Situationally Interesting Contexts

As it became clear that Computer Science had a serious image problem, work began on making Computer Science "Fun" and approachable. A key goal was to increase diversity and to broaden participation. This led to the rise of Situationally Interesting Contexts, emphasizing problems and projects that would be immediately appealing to a wide audience. Guzdial, for instance, was largely responsible for the creation of the Media Computation approach, where students use computational techniques (e.g., iteration and decision) to manipulate sound, images, videos, and other digital artifacts. As an example, students might use a nested, numerically-indexed `for` loop in order to adjust the red-value of the pixels in an image, treating it as a two-dimensional array of binary tuples, in order to reduce the red-eye of a photo.

Although wildly deployed, a review of these curricular materials by Guzdial [28] in light of Situated Learning Theory found that students did not find this an authentic context, and intense rhetoric was insufficient to convince them that it was authentic. Few students find it expedient and helpful to remove the red-eye from family photos by writing python scripts, when they could easily use a GUI-based program to automate the task instead. Guzdial leaves open the question of what contexts can be truly authentic for non-majors, given the relative novelty of teaching introductory computing for non-majors. Ben-Ari echos this question by suggesting a very narrow selection of authentic contexts and communities in his paper exploring the application of Situated Learning Theory to Computer Science in general [7]. Critically, the opposite problem could occur – if an instructor is effective at convincing students a context is authentic, they may believe them. There are serious ethical issues involved in mispresenting the utility of a context, leading students to develop an embarrassing misconception of the field – imagine a young child believing that all of Computer Science is game design, because that is what they started off doing.

There are other disadvantages of an Interest-driven approach. The motivation literature describes "Seductive Details" (interesting but irrelevant adjuncts) [30] as interfering both with short-term problem completion and long-term transfer. In other words, students get hung up on unimportant aspects of the context that they ignore the content. Consider a student using the game and animation development environment Scratch, which allows beginners to create sprites from images. A young learner may be so amused by the ability to change the color and shape of their image, that they neglect their assigned work. Although a well-regulated learner would not be distracted, most of the at-risk population that would benefit from these contexts are unable to deal with such distractions. Of course, this could be said of any context, but there is particular danger from this kind of context.

Kay [37] identifies another, potentially critical problem of relying soley on situationally interesting contexts, particularly when it leads to individualized interest towards that context and not the content. What happens once a student has completed the introductory course and is ready to move onto further course? Most later courses are more decontextualized, and will not use contexts such as game development, robots, etc. Kay goes so far to say that it is unethical to suggest to students that a contextualized introductory course is representative of the curriculum as a whole.

4.2.3 Empowered Contexts

Orthogonal to the idea of making a context fun is the idea of giving students more freedom and agency to control their learning. Compared to Interest-driven contexts, this approach is comparatively less researched, although it is not an uncommon practice – instructors will often allow students to choose from a range of projects or assignments. Stone [55] ran a 2-year study where students were allowed to choose their projects from a wide range of domain areas (e.g., Biology, Math, Business, Etymology), and were then surveyed on their engagement. Unfortunately, their experiment suffered strongly from low enrollments and even lower survey responses; it is difficult to believe any of their results (including the idea that women are more likely to prefer biological- and meterological-themed projects). However, their experiences do suggest an interesting challenge: normalizing the difficulty (both in terms of computational knowledge and domain knowledge) across many different projects is a struggle.

4.2.4 Contexts that Make Instructors Care

Most modern educational theories argue that learning is inescapably affected by social factors. There is evidence that the instructor [57] and fellow students [3] are the most important factors in an introductory experience, for instance. Kay [37] discusses this explicitly. It is possible that the most important element in a context is not whether it is fun or useful, but whether the instructor can get excited about it and impart that enthusiasm to the student. And not just enthusiasm, but a thorough understanding of the problem, its usefulness, and the rest of its attributes.

4.2.5 Useful-Driven Contexts

An alternative focus to Interest is Usefulness, the idea that the context should have immediate or eventual benefit to the learner's needs. To some extent, it is impossible (or at least prohibitively difficult) to find a one-size-fits-all context that will be useful to all learners (doing so is probably an example of preauthentication [49], or designing learning experiences without your learners in mind). The ideal situation for any instructor is to create contexts that specifically suit the interests and values of your learners [19]. However, in practice, some contexts are broadly useful and are likely to engage a diverse crowd of learners. In this section, I suggest two distinct contexts that might fall into this role: real-world problem solving and data science.

In theory, Computer Science provides tools for solving problems, and it is possible that the problem solving can be done with even the simplest tools [27, 40]. Many authors collaborated to produce a new framework centered around these ideas – "Social Computing for Good", a collection of approaches and projects for interdisciplinary students to solve using computing [27]. They raise a number of issues with using socially relevant materials: that games and graphics can be more appeal to instructors as a "cheap" source of motivation, that students and instructors can become cognitively overloaded by the addition of domain knowledge, and that instructors can even be intimidated if they don't have expertise in the domain area. They also create a valuable rubric for developing and evaluating problems (which I map to elements of the MUSIC model below):

1. The degree to which the problem is student-directed (eMpowerment)

2. The amount of scaffolding needed (Success)

3. The amount of external domain knowledge needed (Success)

4. The contribution to the Social Good (Usefulness, Caring)

5. The "coolness" or "sexiness" (Interest)

6. The amount of explicit student reflection incorporated (Usefulness)

Although this framework presents some ideas, there are still unsolved technical and pedagogical problems in how to optimally bring these materials to learners; their paper ends by raising questions about the effectiveness of this approach compared to existing methods.

A number of other researchers have created course materials, with varying degrees of evaluation. Erkan [23] had a sustainability themed curriculum – student surveys completed afterwards suggested some level of effectiveness, although the sample size (N=16) was far too small. In addition to providing two case studies, Buckley [9] suggests an interesting delineation between Social problems (uppercase S, indicating problems general to society) and social problems (lowercase S, indicating problems of personal interest to the learner).

Barker ran a large survey asking why students persist towards majoring in CS [3] (N=113, only freshmen). Critically, they found that Meaningful/Relevant Assignments (subsuming both Interest and Social Usefulness) was a major factor in whether students would persist in the major – however, it wasn't one of the top ones. Interestingly, whether students felt that their workload and pace was appropriate was a much bigger source of importance, suggesting that care and attention should be given to making a context suitably difficult before it is made interesting and useful. This focus on ensuring normalized, appropriate difficulty is echoed by several other authors, all of whom suggest that doing so is not trivial [51, 54].

In the past two decades, the field of Data Science has emerged at the intersection of Computer Science, Statistics, Mathematics, and a number of other fields. This field is concerned with answering real-world problems through data abstractions, and offer a less socially-conscious path to Usefulness. As a context, there are pedagogical penalties for using it, since it introduces a wide variety of new content – visualization, statistics, ethics, social impacts... the list is long [1]. However, a good instructor can downplay the focus on these side-areas as needed, or even emphasize subject matter's strengths (e.g., a statistics major might find it interesting to use their mathematical background to strengthen their problem-solving investigation). However, there are other difficulties: bringing in messy data requires real sophistication by the instructor, especially when working with Big Data.

The use of data analysis as a form of contextualization is not novel, and represents a new and actively growing movement where instructors create programming assignments with specific datasets in mind [2, 18, 29, 56]. Upper division courses have employed these situated learning experiences using data of varying size and complexity for several years [22, 58, 59]. However, in all of these research papers, there is typically very little evaluation of the advantages and disadvantages of using data science in introductory education. Although Sullivan [56] did conduct a study on the difficulty and usefulness of the datasets they provided, they do not go very far in identifying lasting lessons for educators creating such datasets. Other researchers conducted

even less impressive studies: DePasquale [18] included exactly ONE student response in their evaluation.

4.3 Datasets as a Holistic Motivating Context

I propose using datasets as a motivating introductory computing context. As established in the previous subsection, this is not a novel approach – upper forms have always historically used datasets and recently lower forms have started too. However, there is very little evaluation of the success of this approach, and numerous roadblocks exist in using this context. For my dissertation, I will investigate what technology and pedagogy can be used to take full advantage of this context according to a number of constraints. Before describing what I have done and will do, however, this section will describe this context in more detail.

Data science is the process of answering questions by building, exploring, and processing datasets. There are many theoretical models that define the term more strictly, but in general it is described as an iterative model of collection, sanitizing, processing, rendering, and interpreting. Figure 3 gives a visual overview of this process. My research will not attempt to narrowly define data science; instead, the goal is simply to use elements of the data science process.

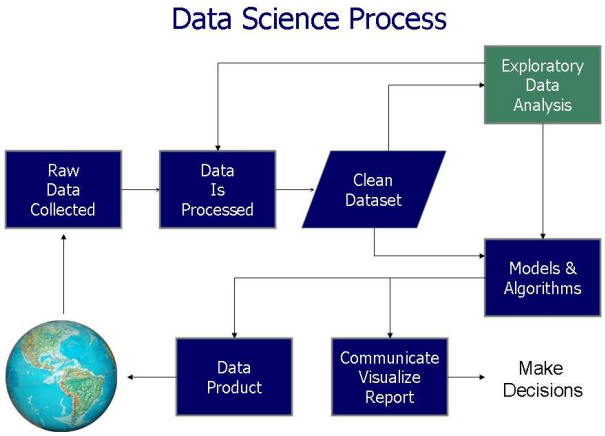


Figure 3. A generalized model of Data Science [14]

It is crucial to understand that the goal is not to teach students how to become data scientists, anymore than it is the goal of Media Computation to teach students how to be professional computational artists. As a context, using datasets is simply a means to an end – it may involve any components of the generalized data science process at any level. Some professors may identify data science as a learning objective in and of itself – that is fine, but they are attempting to teach something that is, at some level, distinct from computing itself.

My core thesis is that data science, used properly, is an excellent context for motivating students across all possible factors of motivation: providing opportunities for agency, a sense of usefulness and authenticity, controllable complexity, the potential to be interesting situationally and towards individualized interest, and a promotion of caring and ethics. Data Science accomplishes this by being an almost "meta-context" – it is a general framework for working with an infinitely wide array of different data contexts. However, there are many challenges in doing so.

4.4 Big Data

A large categorization of data is Big Data. Big Data is much in the news these days, from reports of massive data dredging by the NSA to tens of millions of credit cards stolen by hackers from commercial databases. Big Data has become crucial to scientific advances from understanding

the genome to predicting climate change. Even more so than regular data science, there are many obstacles to effectively educating students on Big Data. Its representation, manipulation, and expression is, by definition, challenging, and modern curriculum and programming tools are typically inadequate.

Big data has been loosely described as quantities of information that cannot be handled with traditional methods [41]. But "traditional methods" is a vague phrase that has different meanings to different learners. To a Humanities major in their first CS-0 course, the traditional method to sum a list is to use Excel. In this scenario, "big data" means anything that won't comfortably fit into Excel's working memory. However, to a third-year Computer Science major, the traditional method would be to write an iterative or recursive sequential loop; being given big data forces them to explore parallel models of execution. Clearly, "bigness" is a function of the learner's experience, but that is still not a solid definition. A more precise definition is the "3V Model" [20], which posits that there are three dimensions that distinguish big data from ordinary, run-of-the-mill data:

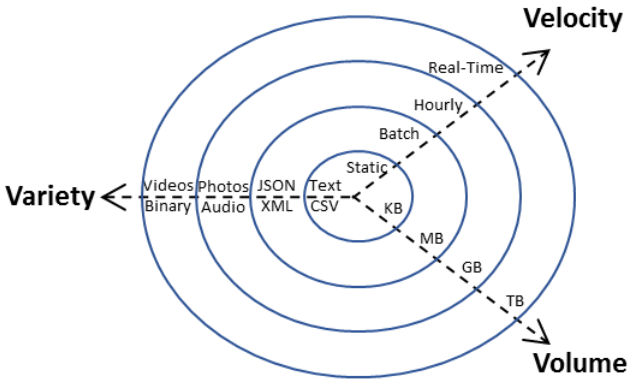


Figure 4. The 3V Model of Big Data

Volume: The total quantity of the information, usually measured in bytes or number of records. However, this also extends laterally: the number of fields in the structure of the data also impacts the complexity and size. The threshold at which data becomes big is a function of the hardware and software being used – for instance, embedded systems may consider gigabyte-sized files to be big, while modern servers might not struggle until the petabyte level.

Velocity: The rate at which new information is added to the system. High velocity big data implies a distributed architecture, since new data must be arriving from somewhere. The dynamicity of data can vary widely across these architectures, with data updating every year, every day, or even multiple times a second.

Variety: The format or formats of the data. Ideally, data are always distributed in a way that is readily accessible – for instance, simple text-based formats such as CSV and JSON are widely supported, relatively lightweight, and human-readable. More sophisticated data formats for image and audio are also typically well-supported, although still more complicated. However, projects using specialized, compressed binary formats or, more dangerously, multiple formats (e.g., image archives organized with XML files), are more complex.

Silva [53] taught an introductory course truly focused on techniques for tackling Big Data: NoSQL, MapReduce, NewSQL. Unfortunately, they did not conduct any kind of evaluation of their work across any of the expected dimensions. Learning to work with Big Data can add extra authenticity to the context, but it also raises a large number of new challenges. Once again, it is

not the goal of my research to teach students how to work with truly Big datasets. Instead, it is simply a means to motivate students. In the following sections, I describe some specific challenges of working with Big Data, and the possible approaches to solving them.

4.4.1 High Velocity Data

It is not trivial to enable introductory students to work with high velocity data, which is necessarily distributed. Without any scaffolding, it is necessary to delay the use of such data until much later in the course. In a prior paper [4], we outline the biggest barriers to high velocity data as a context:

Access The process of programmatically downloading and parsing a web-based resource is a non-trivial procedure requiring an understanding of both basic concepts (e.g., function calls, data transformation) and specialized web technology (e.g., the difference between GET and POST calls, building query parameters).

Non-Idempotency Because high velocity data is constantly changing, repeated calls to the same URL endpoint can return wildly different results, even over the course of a few minutes. This makes finding errors and testing considerably harder.

Consistency Web-based APIs are controlled and developed by independent entities, which means that changes can occur at any time with little to no notification or time for reaction. This means that students' code can become out of date even during the middle of testing their final project.

Connectivity Although internet speeds for students on a university campus are typically stable, this does not extend to off-campus students or students that are traveling. If the internet connection is down, then students might be completely unable to make progress.

Efficiency Even when the internet connection is stable, it might not always be fast. Requiring a round-trip to a server can greatly drag on the testing and development process, frustrating the student and decreasing the time spent learning.

4.4.2 High Volume Data

Data Transmission: Internet connections can be difficult and inconsistent, especially for off-campus and non-traditional students. Although most modern universities boast impressive wired connection speeds, these speeds rarely extend off-campus. And even when internet connections are top-notch, they can still be inadequate to serving the needs of transmitting big data collections to an entire classroom of students. Some affordances must be made to make the data readily available to students without taxing their hard drives unnecessarily.

Historically-oriented data: Additionally, high volume data offers different contexts and problems than high velocity data. For instance, high velocity data typically lends itself to small quantities of data that are relevant to the current state of the real world – for instance, students can walk outside and feel the current weather, which should correlate to real-time weather reports made available by a weather library. High volume data, on the other hand,

lends itself to large quantities of mostly static data – for instance, crime reports for a long period of time. Although high velocity data gives authentic answers in the here and now, high volume data gives authentic answers for the future, through trends. Some fields have both kinds of data available – meteorologists generate forecasts (high velocity, low volume) by studying historical climate data (high volume, low velocity). But some fields are not amenable to both – digital historians typically have large stores of historical information (high volume), but it does not change quickly (low velocity). Careful consideration must be made when choosing problems and designing contexts so that the data leads to optimally authentic learning experiences.

4.4.3 High Variety Data

Inconsistency of Storage: High variety data is composed of many different kinds of file formats – some of which are more complicated than others.

Inconsistency of Tools: Different languages usually offer many different tools to interact with the exotic formats of the data – however, these tools vary greatly in availability, usabililty, and compatability across platforms. For instance, binary image data is supported as a first-class programming object in the Racket programming environment, but can only be loaded using libraries such as Pygame in Python, which the user may or may not have installed.

5 Existing work

As I am now three years into my PhD, my research builds on significant prior work. First and foremost, my CORGIS project has already been used in several introductory programming experiences, and seen publication at multiple venues, including two very successful workshops. My primary research project for the first two years of my dissertation led to CORGIS: Collections of Real-time, Giant, Interesting, Situated Datasets. This project's goal is to make simplistic data sources available to learners early in their programming experience, so that they can explore Big Data Science contexts.

5.1 Technical Infrastructure

The foundation of this proposal rests on prior work developing the RealTimeWeb project, a software architecture framework that provides introductory programming students with an easy way to access and manipulate distributed real-time data [5]. Real-time data is a specific branch of Big Data – specifically, high velocity data.

As our focus shifts from real-time data to datasets in general, we have renamed our overarching project from RealTimeWeb to CORGIS. Our work is now available at `http://think.cs.vt.edu/corgis/`. As a successor to the RealTimeWeb project, CORGIS retains all the previously developed libraries for accessing high velocity data; however, it also contains our new libraries for working with high volume data. These libraries are paired with potential assignments and helpful documentation for deployment. Long term, we intend to gather and disseminate data on the success of these libraries, with the hopes to establish a community of developers and educators that create new

resources. For clarity's sake, we use the name RealTimeWeb to refer to the architecture we have developed to rapidly connect to high velocity data streams.

At the heart of our project are carefully engineered, open-source client libraries through which students can access the data provided by real-time web services. We provide client libraries for a number of data sources, such as business reviews from Yelp, weather forecasts from the National Weather Service, and social content from link-sharing site Reddit.com. Each of these client libraries is in turn available for three common beginner languages: Java, Python, and Racket. These libraries do more than just streamline the process of accessing distributed data, however; each library is built with a persistence layer that enables the library to work without an internet connection. Not only does this ensure that students without a solid internet connection can maintain productivity, it also simplifies developing unit tests. In fact, this technical scaffolding for the students circumvents most of the difficulties of distributed computing, including HTTP access, data validation, and result parsing. Figure 5 demonstrates the architecture used in our libraries.

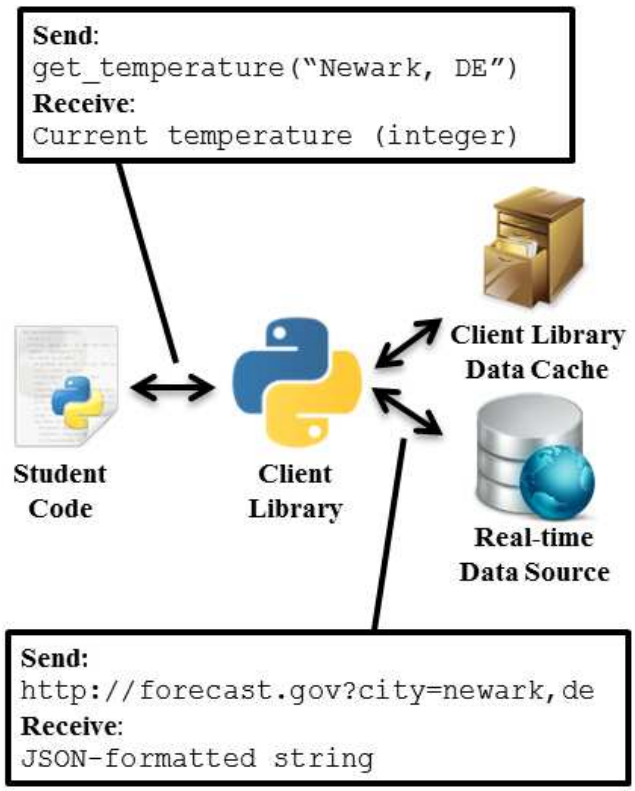


Figure 5. RealTimeWeb Client Library Architecture

The persistence layer is implemented using a caching mechanism, but not a traditional one. In a conventional caching system, the result of every call made to the external service is memoized using a key-value store, often with a timestamp in order to expire out-of-date data. In our caching system, an instructor preloads the cache with a sequence of data values for each expected call to the external service. Although this limits the number of possible calls to the external service, it improves the consistency of the experience. Instructors can specify policies for how the system returns data – if the cache runs out, it could restart with the initial result, a developer-specified "empty" result, or repeatedly return the final result. For example, consider the United States Geological Services' Earthquake data stream, which exposes a function to retrieve a list of earthquakes around the world for a given time period (e.g., past hour, past day, past week, etc.). The instructor could provide a cache to simulate a period of high seismic activity, returning a large number of earthquakes every time the function is called. If the user exhausts the data in the cache, it might be programmed to return an empty list, signaling no further activity.

Our High Volume libraries often work by providing sampled versions of the data internally, so that students can work with faster, representative subsets. Then, when they are ready for full deployment, they can switch to a "full production" mode – trading speed for quality. Most of the work in developing these libraries is organizing the sampled data to load into memory and get

processed as quickly and naturally as possible.

An alternative scheme for distributed High Volume libraries uses a more traditional caching strategy – assuming that the data source is largely static, requests are cached locally. This caching is typically done using a simple key-value store on the local filesystem. Limits can be set on the size or lifespan of the data in the cache, allowing the system to update itself according to the velocity of the external data service. Using open-source REST API generating tools such as Eve [32], existing high volume datasets can be quickly transformed into distributed datasets, alleviating the issues of data storage and transmission.

5.2 Semi-Automatic Library Generation

Our client libraries are easily available through a curated, online gallery; each library is designed to be quickly adapted to instructors' specific academic desires. This gallery also provides a tool for rapidly prototyping new libraries based on our framework. As an open-source project, we encourage collaborators to explore and extend the tools that we have created.

The process of connecting to online data sources is fairly uniform: performing an HTTP request to a URL returns formatted data (typically JSON or XML). This information must then be parsed into native data structures, and then filtered and transformed into an appropriate domain object using the language's proper construct (e.g., structs for Racket, classes for Java). We've established a JSON-based client library specification file format that can be compiled to appropriate source code in each of the three target languages using the modern templating language Jinja2. This compiler can be easily extended to new languages by providing a template. This specification format has two major components: the functions, which connect to the relevant URL endpoints, and the domain objects, that are generated from a successful HTTP request. This open-source tool has been used to successfully and rapidly develop a number of the existing client libraries.

5.3 Contextualizing with CORGIS

The CORGIS project has been deployed for several semesters in introductory Computer Science courses for majors, ranging from the first course all the way to a Data Structures level course. These integrations ranged from small assignments to entire semester projects using the software. So far, the focus of the evaluation has been on the motivational influence of the system. Quantitative data was collected by surveying students attitudes using well-established motivational frameworks and instruments, and indicates that students tended to find real-time data engaging [6]. In some courses, qualitative data was gathered through small group interviews, where students attribute increased engagement with the authentic, real-world connection offered by real-time data.

Similar results have been found from its integration in a Computational Thinking course – students cite working with the realistic data as a key factor in engaging with course materials. Refer to figure 6, the result of an end-of-course survey (34 responses, 87% response rate). Students were asked to respond via a Likert scale to the statement, "Other schools teach Computational Thinking in different ways. Consider these different styles, and indicate how much you would prefer or avoid them." Although this is an extremely shallow question, since students were only exposed to two contexts (computational modelling and working with data), it is an encouraging, preliminary result.

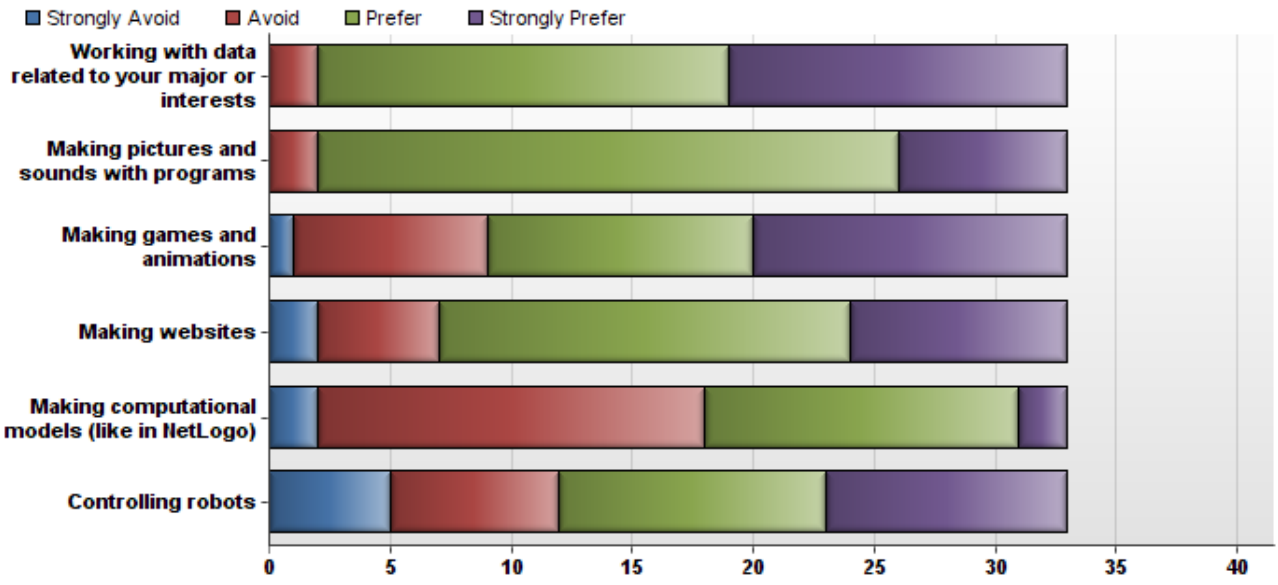


Figure 6. CT Students' Perceptions of Various Introductory Programming Contexts

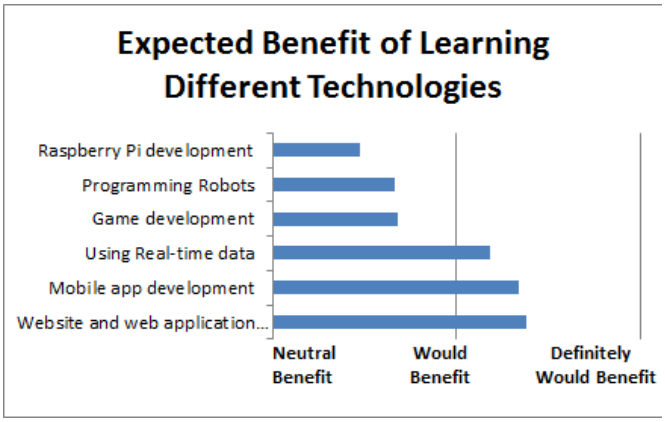


Figure 7. CS Majors' Perceptions of the Benefit of Learning Different Technologies

5.4 Block-Based Programming

Programming is a difficult thing to learn, and there exist many tools to scaffold that process. One of the more popular approaches is Block-based Programming Environments, which allow the user to build their program from pieces of provided code constructs. This prevents common syntax errors and some of the the "blank canvas" effect. The CORGIS project has been leveraged in several different block-based environments, including a Python-based system based on Blockly and the popular programming environment/language Snap!. Both systems implement their own layer around CORGIS in order to provide access to rich datasets. The Python/Blockly system has additional tools for creating plotted visualizations of data, and the Snap! interface has generalized support for accessing online data. Although the Snap! tools were created as part of a spin-off project, the Python/Blockly tools are my own work.

During the Computational Thinking course, students take advantage of the internal caching mechanism of the CORGIS libraries to work on recorded data. This simplifies the debugging process since programs perform predictably and are no longer dependent on external data services. When students are ready to run their programs against live data, they can move offline to a traditional programming environment and run in regular production. This caching mechanism also benefits from an assessment view - a students program can be checked for robustness by invisibly changing the values returned by the big data functions. Figure 8 demonstrate the CORGIS library in action both in regular Python code and the equivalent Blockly code.

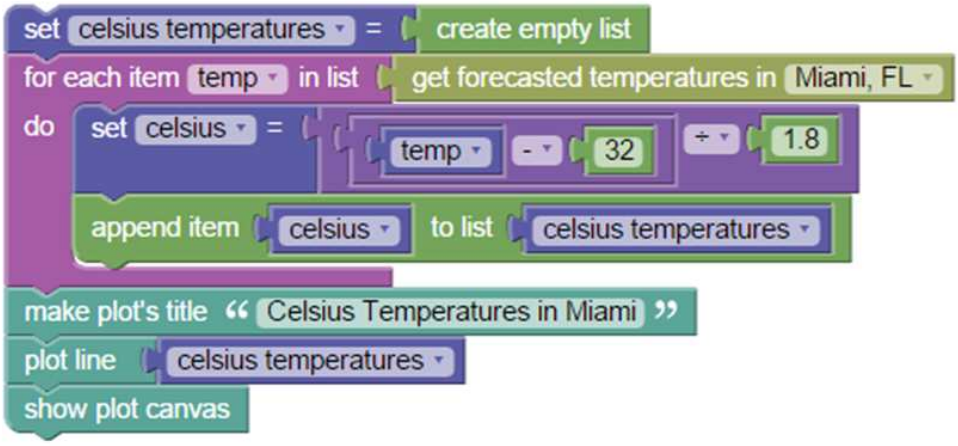
6 Intervention Context

With the development of new technology described in the following section, interventions will be staged through Virginia Tech's new "Introduction to Computational Thinking" course, created to help fulfill the university's new General Education Requirements [45]. This course has already been run for two semesters, deeply incorporating much of my existing research work. The course is taught and developed primarily by Dr. Dennis Kafura, although I have also been involved as associate instructor, managing course materials, server and technology administration, and assisting with in-class teaching. However, now that the course is more solidly defined, my role is shifting into a more observational function in order to drive my dissertation work. As a research endeavor, the course is heavily instrumented to provide data on its novel pedagogies and technologies. Although there are confounding factors to working with such a heavily experimental course, it presents a unique testbed for materials and is an excellent source for mining research results.

6.1 The Learners

The students in the Computational Thinking course present a unique profile. A few of them will have had prior programming experience, but most of them have had very minimal interactions with computers (indeed, they often describe themselves as "not a computer person"). These students may not believe that Computational Thinking will help them. This is largely because they have more clearly defined domain identities, and may not see how Computational Thinking fits into them. So, indeed, these students often have low motivation, especially in their sense of Success and Usefulness.

Figure 8. Blockly vs. Python Code




```
import weather
import matplotlib.pyplot as plt

celsius_temperatures = []
for t in weather.get_temperatures("Miami, FL"):
        celsius = (t - 32) / 1.8
        celsius_temperatures.append(celsius)
plt.title("Celsius Temperatures in Miami")
plt.plot(celsius_temperatures)
plt.show()
```

Gender

	Fall 2014	Spring 2015
Female	6	21
Male	13	18

Prior Programming Experience

	Fall 2014	Spring 2015
Yes	10	7
No	10	32

Year

	Fall 2014	Spring 2015
Freshman	2	5
Sophomore	7	11
Junior	6	11
Senior	5	10
Unknown	0	2

Colleges

	Fall 2014	Spring 2015
Engineering	2	2
Agriculture	0	1
Sciences	7	5
Liberal Arts	9	23
Architecture	1	7
Natural Resources	0	0

Figure 9. Demographic Data of Computational Thinking Students

In the first offering of the course, 25 students enrolled in the course, and 20 students finished the coursework. In the second offering, 40 students initially enrolled and 35 successfully passed the course. Figure 9 indicates the relevant demographic data collected through surveys. Largely, the students represent the population at Virginia Tech, albeit with some disproportion towards certain majors than others. It is worth pointing out the excellent gender diversity within the class.

6.2 The Content

Virginia Tech defines four learning objectives for "Computational and Quantitative Thinking" [45]. Although the Computational Thinking only satisfies four of these objectives outright, they are all considered valuable end-goals:

1. Explain the application of computational or quantitative thinking across multiple knowledge domains.
2. Apply the foundational principles of computational or quantitative thinking to frame a question and devise a solution in a particular field of study.
3. Identify the impacts of computing and information technology on humanity.
4. Construct a model based on computational methods to analyze complex or large-scale phenomenon.
5. Draw valid quantitative inferences about situations characterized by inherent uncertainty.
6. Evaluate conclusions drawn from or decisions based on quantitative data.

This content is mapped roughly into four instructional units on Computational Modelling, Algorithms, Data Intensive Inquiry, and Social Impacts. The lattermost unit is threaded throughout the course, while the first three are roughly sequential. Figure 10 gives a high-level overview of the content of this course.

It is clear that this material aligns smoothly with the content described in this preliminary proposal, in particular a focus on algorithms and abstraction.

Topic (Length)	Description
Computational Modeling(2 weeks)	Model-based investigation of how complex global behavior arises from the interaction of many agents, each operating according to local rules. Students use case-based reasoning and encounter basic computation constructs in a highly supportive simulation environment.
Fundamentals of Algorithms(4 weeks)	Study of the basic constructs of programming logic (sequence, decisions, and iteration) and program organization (procedures). A block-based programming language is used to avoid syntactic details. Students can see how these constructs are expressed in Python.
Data-intensive Inquiry(7 weeks)	Project-based exploration of complex phenomena by algorithmically manipulating large-scale data from real-world sources. Students construct algorithms in Python using a supportive framework for accessing the data.
Social Impacts(2 weeks)	Explore and discuss contemporary societal issues involving computing and information technology.

Figure 10. High-Level Course Overview

6.3 The Course

The course uses a considerable amount of modern pedagogical techniques, many of which represent ongoing research questions. Perhaps the most influential technique is the organization of students into cohorts. Near the beginning of the semester, students are put into groups of 5-6, balancing based on year and gender where possible, and avoiding putting similar majors into the same group. These cohorts primarily function as a support structure that students can rely on to get help and encouragement. Although cohorts work together on many smaller in-class assignments, every student is ultimately responsible for their own work – the final project, for instance, is individual to each student.

Class time is split between presentation (typically stand-up lecture) and participation (typically computer-based work or cohort discussion) using an Active Learning style whereever possible. Earlier questions in the course often have students completing questions on paper or doing more kinetic exercises. Later questions rely on the automated Kennel questions, until the students reach the open-ended project work.

Work in the class is considered to be under a mastery style – deadlines are enforced only in so far as they motivate students to complete assignments, not in order to punish. Students are free to work on the material as long as they need. A recurring message within the course is that "failing is okay, as long as you keep trying".

7 Research Questions

In this section, I outline the research problems that I will explore.

7.1 The Use of Datasets

What are the issues using datasets, and how do we manage them?

In this section, I outline a number of the biggest issues in using datasets as an introductory context. Existing research in this area is largely concerned with providing examples of how to use datasets, rather than solving these more generalized problems. I will develop meaningful classifications, theories, guides, and materials to help instructors and course developers manage these issues.

7.1.1 Preparation of Datasets

One of the most general questions when it comes to these datasets is how to organize and prepare the dataset. Where should the data come from? Data can be collected by organizations, by the instructor, by the students (e.g., using data in their own lives), or any of a number of different sources. That data can be based on a simulation or predicted model (e.g., fantasy football), or observations of a real phenomenon (e.g., weather reports)? I will analyze these different sources, the affordances and costs associated with them, and develop conclusions on the best way to leverage them. This will include guides on how to find and develop datasets.

7.1.2 Normalization of Difficulty

An incredibly useful problem to solve is how to meaningfully normalize difficulty between datasets and identify the difficulty of a given dataset. If this could be accomplished, instructors could confidently empower students to pursue distinct projects while ensuring that they are having a mostly uniform experiences. Difficulty creeps through in a number of ways, as described in the following list.

Domain Knowledge Required: Understanding a dataset requires the student to understand the greater context, entities, relationships, and reality that the dataset abstracts. In some cases, students (especially in Computational Thinking) bring useful prior knowledge that help them to rapidly comprehend a dataset. When this knowledge is not present, an extra level of difficulty is added. For example, in the Computational Thinking class, students easily grasped the Weather dataset (featuring an abstraction of temperature as a number), but struggled with the Stock Value dataset (featuring an abstraction of stock values as numbers) – even though both datasets had the exact same data structure.

However, there are other ways that domain knowledge can be a problem – students' prior knowledge can be limited, incorrect, or difficult to leverage, for example. I will describe best practices for helping students get to know datasets and identifying whether a dataset is suitable for a given student or sets of students.

Computing Knowledge Required: Anyone making a dataset must become familiar with how to structure the data. Anyone using a dataset must become familiar with how to process it. Any programming interfaces made available (e.g., special libraries and frameworks) incur their own costs. Although some languages are made specifically to handle data processing using high-level declarative techniques (e.g., SQL), other languages require different levels of

specificity on how to manipulate the data. In a practical setting, there are major advantages to choosing the "simplest" option (one might argue, for example, that a declarative language makes filtering easier than a functional language does, and a functional language makes filtering easier than a imperative language does). However, when the goal is to teach concepts, there may be other advantages related to the learning objectives to using these datasets. I will analyze and describe the pedagogical affordances that different data structures and programming styles create.

Included in this analysis is research on the effectiveness of the Block-based Programming Environment for guiding students to work with datasets. As previously mentioned, the Block-based environment used in the Computational Thinking course has first-order support for integrating datasets. However, there are open questions as to how effective the environment is in teaching students. I will expand and evaluate the Block-based Environment's capabilities for supporting students understanding of working with datasets and the introductory computing content.

Dataset Knowledge Required: In general, learning to work with a new dataset requires time and attention to become familiar with its particular structure and design. Why was it organized the way it was (e.g., as a list of dictionaries as opposed to a dictionary of lists).

Understanding the structure, meaning, and values of data can be a difficult problem for introductory students. Using Data Science as the introductory context puts an increased emphasis on data structures and algorithms, making this content more critical for the learner. Currently, there are not many tools available to scaffold students in getting to know datasets. An example of a possible approach is Spyder's Variable Explorer, seen in figure 11, which allows users to drill-down through a data structure (assuming it uses native Python `lists` and `dicts`) by opening nested windows. Although this tool is powerful, it is limited pedagogically by offering very little supplementing knowledge. However, it does provide a useful example of a way to visually explore a dataset.

A number of graphical representations are used throughout the Computational Thinking course in order to visualize topics such as Abstraction and program flow. For instance, students build Data Map Diagrams as seen in Figure 12, intended to help them navigate complex data structures. A system will be created to generate simplified graphical visualizations of the students code and the data model in order to connect to these other course topics. The system will be available for data used in the course regularly, and should be extremely beginner-friendly.

On the other hand, guidance will be created for dataset developers and instructors to adjust the structure and content of a dataset to normalize its difficulty. For example, a highly-nested dataset can be semi-automatically adjusted to drop levels of nesting in favor of a flatter structure. Alternatively, a list datapoints could be grouped by one of their features to make it easier to conduct grouped analysis (e.g., grouping weather reports by city or year so it is easier to calculate averages). There are a wide range of techniques that could be suggested based on a situation – automatic analysis of the dataset could help provide suggestions on where the complexity can be increased, decreased, or aligned with specific learning objectives.

7.1.3 Reliability of the Dataset

There are a number of issues that the student and instructor should become aware of about the nature of the data, as a secondary concern. Was the data collected correctly? Were mistakes made in processing it? Are the abstractions false? Although some instructors may not find it a primary learning objective, students need to be aware of the possibilty of mistakes in the process of preparing a dataset, and instructors need to be aware of the ethical ramifications of presenting a dataset as reliable or authentic.

Careful consideration must be made when choosing problems and designing contexts so that the data leads to optimally authentic learning experiences. In practice, datasets can vary greatly in authenticity – some data is collected incorrectly or has other errors, some data was predicted from a model rather than observed from real phenomenon. A curious component of authenticity, however, is that it is a function of the observer. A persuasive instructor might convince a class of students that an entirely artificial dataset was representative of real-world data, especially if it confirmed students' existing biases. There are ethical issues with artificial datasets and the stories that they tell. However, there are serious pedagogical benefits to generating datasets that fit instructor's goals – data that leads to interesting visualizations, or clean results. An important facet of my research will be exploring the ethical and pedagogical ramifications of the authenticity of datasets.

One of the big dangers when attempting to create meaningful context for learners is the problem of *Preauthentication*: attempting to design for authenticity without sufficient knowledge of the audience. This is a problem shared by any approach to introductory material. Petraglia gives a compelling example [49]:

> The task of balancing a checkbook, for instance, may be an authentic task from the perspective of a 21-year-old, but we would question its authenticity from the perspective of a 5-year-old. But more to the point, even among 21-year-olds, for whom we believe the task should be authentic, there are some who will find any given lesson in personal finance irrelevant, inaccurate, or otherwise inappropriate.

Preauthentication stems from over-generalizations and run-away assumptions. If you attempt to reduce an entire classroom to a list of likes and dislikes, you run the risk of ignoring each individual learner's rich history and background that they will be building from. It is difficult to plan for and work against this ever-present danger when designing reusable assignments. Petraglia [49] recommends that rather than attempting to design around students prior understanding, it is better to simply convince the learner of the authenticity of the problem. But this is limiting, since it ignores the prior experiences and understanding that a student brings to their learning. Instead, it would be better to find a middle ground where students are given flexibility while maintaining a relatively uniform experience for students. In an ideal learning environment, students will have freedom to explore datasets of their own choosing, possibly from a list. Of course, this must be balanced with the students inexperience with finding datasets, requiring the process be given time and attention.

7.1.4 Availability of the Dataset

Some organizations make their data available only through certain channels – students have to sign a legal agreement, or register for an API key. Organizations such as FERPA and HIPPA exist in order to ensure the privacy and dignity of captive populations (e.g., school children, patients). These organizations define rules for how data on such populations can be published and made available. Often, interesting data exists behind walled gardens. Although novel techniques such as Differential Privacy (where data is probabilistically modified to protect anonymity [21]) can be used to mitigate these problems, it is simply a fact that some data is utterly unaccessible. I will explore the options available to mitigate the dataset, and their affect on the authenticity of the data.

Of course, the format of the dataset itself may lead to unavailability. Many developers have limited experience, time, and interest with the best way to package data for others' consumption, especially when it comes to beginners. It can be easy to release a dataset as a PDF or in some obscure binary format. Overtime, data can also disappear behind dead/moved URLs. A major challenge for organizing data can be finding it and interpreting it. I will analyze the common pitfalls committed in releasing data to develop preventive and corrective measures.

7.1.5 Other Motivational Value of the Dataset

A given dataset has some motivational value that is a function of the student, the context, and the dataset itself. Although Goldweber et al [26] attempted to define these elements in a general sense, there is no coherent model to holistically describe how motivation is imparted in datasets. I will analyze the factors and elements of a motivation and derive a detailed framework for evaluating the motivational value of a dataset. Examples of the kinds of issues present in this question are:

Interestingness of the Dataset: Does the dataset have generalized appeal, or is it more niche?

Freshness of the Dataset: Has this dataset been explored before? Have people already used it to answer questions?

Usefulness of the Dataset: Does the dataset hold the answers to any questions? Is getting results from it like pulling water from a stone? How much work is required to answer these questions – is it a problet or a project?

Social Nature of the Dataset: Does the dataset relate to a real-world problem? Or is it more esoteric or socially (lowercase S) natured?

7.1.6 Comparison of the Context

As outlined in section 3, there are a number of different contexts available to instructors. What makes data science a particularly effective one, compared to the alternatives? I will conduct surveys of students in order to find out how they compare from their perspective. The goal of this research question is to find out the trade-offs and value of this context, and whether it can be a true all-in-one.

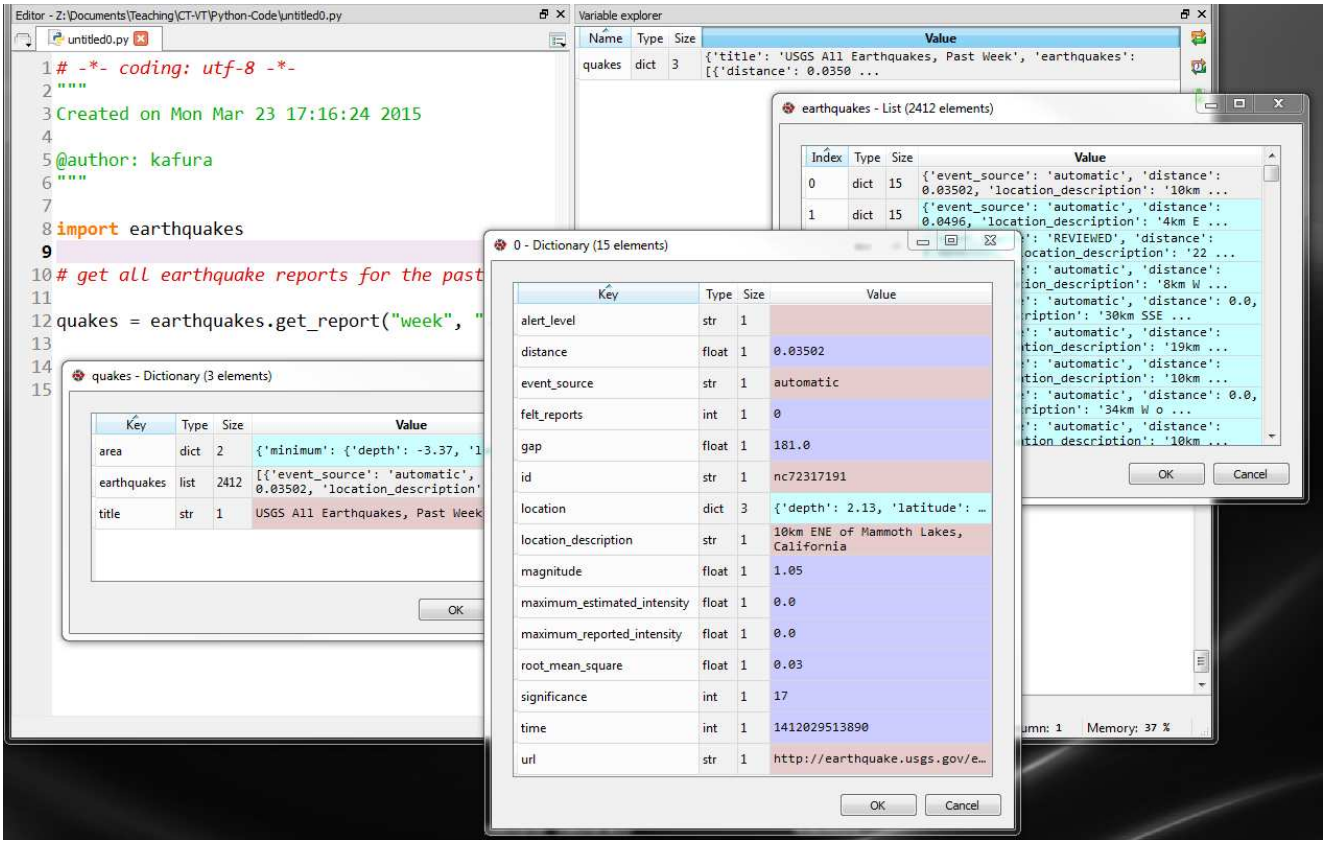


Figure 11. The Variable Explorer in the Spyder Programming Environment

7.2 Academic Motivation and Learning?

There is strong evidence that motivation predicts performance in the course, and my research will contribute to this growing base. Motivation in the course will be primarily assessed through self-reported surveys based heavily on the MUSIC Model of Academic Motivation Inventory (MMAMI). Students will be non-anonymously surveyed using the complete instrument at key points in the semester, and regularly surveyed using a miniaturized version throughout the semester. These miniaturized versions will be used as follow-ups to specific course exercises in order to determine more finely-grained answers about where students found motivation. Additionally, targeted qualitative data will be gathered in order to refine this quantitative data.

This motivational data will be combined with evidence of engagement with the course across several different dimensions. Therefore, the complete list of motivational sources is:

Attendance: Did the students regularly attend class?

Retention: Did the students complete the course with a passing grade?

Coursework: Did the students regularly complete coursework and homework on time?

Struggle: Did the students continue with materials even after setbacks?

Continuation: Do students intend to follow-up the course with further courses and learning experiences in Computational Thinking?

Observations: Did the course staff report students as being motivated, according to their own observations? In Spring 2014, for example, teaching assistants were asked to rank their students on a scale of 1–4 to indicate how engaged they were.

Self-report: Did the students report themselves as being motivated, according to MMAMI and other surveys?

Success: Did the students succeed at the final project and overall course?

This evidence, along with the MUSIC model's data, can be used to explore academic motivation within Computer Science more holistically.

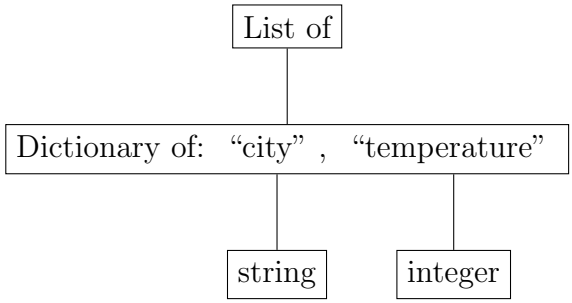
7.3 Extension of Materials

How can we disseminate the research materials and results I am developing?

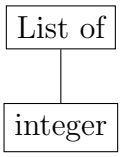
The CORGIS project depends on having an extensive and varied collection of datasets. Students should be able to find a dataset that appeals to their personal interests but provides a sound learning experience. Making this experience relatively uniform is challenging, but makes providing support much easier – a necessary element in a scaled classroom.

The same data source can be expressed in many ways, representing different levels of abstraction and different affordances. Consider the data map in figure 12, representing potential structures for weather data collected about cities around the world. Although maps A and B contain the exact same data, they are structured very differently – a list of cities vs. a dictionary of cities. For an experienced programmer, these differences are minor details that have implications for runtime performance: looking up a city's temperature is slower and more complicated in structure A (requring iteration and decision) than it is in B (requiring only dictionary access). Although the differences in code are minor to an expert, they require fundamentally different areas of knowledge for a beginner. Map C represents an entirely different level of abstraction, where weather is only represented as a numeric, without information about cities. The number of questions that can be answered using this data is greatly reduced – statistics about cities in general, rather than comparisons against specific cities. And of course, the nature and schema of the data itself affects the types of questions that can be asked – it makes sense to find the average of a list of temperatures, but not the sum. A key part of my research will be explicitly identifying trade-offs and affordances of different structures and abstractions of data.

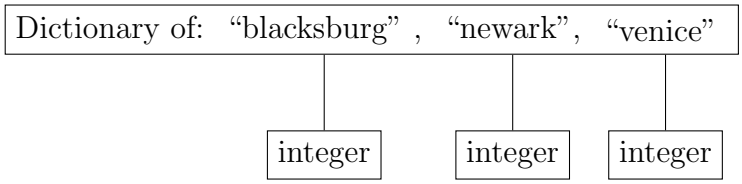
Finding data sources can be a challenging task. A component of my research plan is to publish best practices and techniques for finding, organizing, and exposing datasets. The CORGIS collection already contains three dozen libraries, covering areas as diverse as international studies, nutrition, and social media platforms. Some datasets were chosen because they were low-hanging fruit – extensively documented, freely available, and well-supported. However, other datasets were driven by students' needs; in the two semesters that the course has been offered, two dozen new datasets have been created on demand. The rapid turn-around time required has given new insight into the process of data mangling, although I am still exploring methods. A massive part of my proposed work is to solidify and publish these techniques into a full paper.



(A) List of Weathers



(C) List of Temperatures



(B) Dictionary of Weathers

Figure 12. Weather Data Maps

In addition to guides on the development of datasets, I will create guidance on how to meaningfully integrate these materials at multiple levels of a course. Drawing on principles of Instructional Design, I will create materials that are quickly adaptable to courses at the problem-level and the project-level. Instructors will be provided with guides on how to integrate these datasets into their courses most meaningfully.

8 Work and Publication Plan

This section outlines a schedule for my research over the next two years.

- Spring 2015
  - Second offering of CT course, collect baseline data (N=40)
  - Alpha version of Blockly/Python CORGIS integration
  - Second iteration of CORGIS libraries being used in a course
- Summer 2015
  - Preliminary Proposal
  - TOCE Journal Paper on Big Data Science Affordances
  - Beta version of Blockly/Python CORGIS integration
  - Block-based Language Workshop on Blockly/Python CORGIS integration
- Fall 2015

  - SIGCSE'16 Paper publicly announcing and describing Blockly/Python CORGIS integration
  - Third offering of CT course, collect data (N= expected 60-80)
- Spring 2016
  - Fourth offering of CT course, collect data (N= expected 80)
- Summer 2016
  - Development of second iteration of Blockly/Python CORGIS integration
  - Finalization of all major CORGIS architectures
  - TOCE Journal Paper on the design and issues of Virtual Learning Environments
- Fall 2016
  - Fifth offering of CT course, collect data (N= expected 80)
  - SIGCSE'17 Paper on affective/cognitive student tracking in Kennel
- Spring 2017
  - ITICSE'17 Paper, as a summative view of my primary research questions (using CORGIS to meaningfully motivate and guide large quantities of students to success).
  - Final Thesis Defense

9 Conclusion

My research is focusing on motivating introductory computer science classes with broadly applicable contexts. I seek to introduce new tools and approaches that can support instructors, developers, and learners in using data science and datsets as an introductory context. In this preliminary proposal, I describe the technical work and how it will be deployed and evaluated in a Computational Thinking classroom. I anticipate that this approach to introductory computing will provide useful new techniques and tools instructors. The research results that I uncover will advance the literature in an crucial way.

A Definitions of Computational Thinking

A.1 AP CS Principles

P1: Connecting computing

Developments in computing have far-reaching effects on society and have led to significant innovations. These developments have implications for individuals, society, commercial markets, and innovation. Students in this course study these effects and connections, and they learn to draw connections between different computing concepts. Students are expected to:

- Identify impacts of computing;
- Describe connections between people and computing; and
- Explain connections between computing concepts.

P2: Developing computational artifacts

Computing is a creative discipline in which the creation takes many forms, ranging from remixing digital music and generating animations to developing websites, writing programs, and more. Students in this course engage in the creative aspects of computing by designing and developing interesting computational artifacts, as well as by applying computing techniques to creatively solve problems. Students are expected to:

- Create an artifact with a practical, personal, or societal intent;
- Select appropriate techniques to develop a computational artifact; and
- Use appropriate algorithmic and information-management principles.

P3: Abstracting

Computational thinking requires understanding and applying abstraction at multiple levels ranging from privacy in social networking applications, to logic gates and bits, to the human genome project, and more. Students in this course use abstraction to develop models and simulations of natural and artificial phenomena, use them to make predictions about the world, and analyze their efficacy and validity. Students are expected to:

- Explain how data, information, or knowledge are represented for computational use;
- Explain how abstractions are used in computation or modeling;
- Identify abstractions; and
- Describe modeling in a computational context.

P4: Analyzing problems and artifacts

The results and artifacts of computation, and the computational techniques and strategies that generate them, can be understood both intrinsically for what they are as well as for what they produce. They can also be analyzed and evaluated by applying aesthetic, mathematical, pragmatic, and other criteria. Students in this course design and produce solutions, models, and artifacts, and they evaluate and analyze their own computational work as well as the computational work that others have produced. Students are expected to:

- Evaluate a proposed solution to a problem;
- Locate and correct errors;
- Explain how an artifact functions; and
- Justify appropriateness and correctness.

P5: Communicating

Students in this course describe computation and the impact of technology and computation, explain and justify the design and appropriateness of their computational choices, and analyze and describe both computational artifacts and the results or behaviors of such artifacts. Communication includes written and oral descriptions supported by graphs, visualizations, and computational analysis. Students are expected to:

- Explain the meaning of a result in context;
- Describe computation with accurate and precise language, notation, or visualizations; and
- Summarize the purpose of a computational artifact.

P6: Collaborating

Innovation can occur when people work together or independently. People working collaboratively can often achieve more than individuals working alone. Students in this course collaborate in a number of activities, including investigation of questions using data sets and in the production of computational artifacts. Students are expected to:

- Collaborate with another student in solving a computational problem;
- Collaborate with another student in producing an artifact; and
- Collaborate at a large scale

A.2 CSTA

Computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

1. Formulating problems in a way that enables us to use a computer and other tools to help solve them.
2. Logically organizing and analyzing data
3. Representing data through abstractions such as models and simulations
4. Automating solutions through algorithmic thinking (a series of ordered steps)
5. Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
6. Generalizing and transferring this problem solving process to a wide variety of problems

These skills are supported and enhanced by a number of dispositions or attitudes that are essential dimensions of CT. These dispositions or attitudes include:

1. Confidence in dealing with complexity
2. Persistence in working with difficult problems
3. Tolerance for ambiguity
4. The ability to deal with open ended problems
5. The ability to communicate and work with others to achieve a common goal or solution

A.3 Google Definition

Computational thinking (CT) involves a set of problem-solving skills and techniques that software engineers use to write programs that underlie the computer applications you use such as search, email, and maps. Here are specific techniques: [33]

Decomposition The ability to break down a task into minute details so that we can clearly explain a process to another person or to a computer, or even to just write notes for ourselves. Decomposing a problem frequently leads to pattern recognition and generalization, and thus the ability to design an algorithm.

Pattern Recognition The ability to notice similarities or common differences that will help us make predictions or lead us to shortcuts. Pattern recognition is frequently the basis for solving problems and designing algorithms.

Pattern Generalization and Abstraction The ability to filter out information that is not necessary to solve a certain type of problem and generalize the information that is necessary. Pattern generalization and abstraction allows us to represent an idea or a process in general terms (e.g., variables) so that we can use it to solve other problems that are similar in nature.

Algorithm Design The ability to develop a step-by-step strategy for solving a problem. Algorithm design is often based on the decomposition of a problem and the identification of patterns that help to solve the problem. In computer science as well as in mathematics, algorithms are often written abstractly, utilizing variables in place of specific numbers.

A.4 Cuny, Snyder, Wing

Computational thinking for everyone means being able to

- Understand what aspects of a problem are amenable to computation
- Evaluate the match between computational tools and techniques and a problem
- Understand the limitations and power of computational tools and techniques

- Apply or adapt a computational tool or technique to a new use
- Recognize an opportunity to use computation in a new way
- Apply computational strategies such divide and conquer in any domain

Computational thinking for scientists, engineers, and other professionals further means being able to

- Apply new computational methods to their problems
- Reformulate problems to be amenable to computational strategies
- Discover new science through analysis of large data
- Ask new questions that were not thought of or dared to ask because of scale, easily addressed computationally
- Explain problems and solutions in computational terms

References

[1] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and B. Tribelhorn. A data programming cs1 course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 150–155, New York, NY, USA, 2015. ACM.

[2] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and S. A. Wolfman. Introductory programming meets the real world: Using real problems and data in cs1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 465–466, New York, NY, USA, 2014. ACM.

[3] L. J. Barker, C. McDowell, and K. Kalahar. Exploring factors that influence computer science introductory course students to persist in the major. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 153–157, New York, NY, USA, 2009. ACM.

[4] A. C. Bart, E. Tilevich, S. Hall, T. Allevato, and C. A. Shaffer. Using real-time web data to enrich introductory computer science projects. In *Splash-E '13*, Oct 2013.

[5] A. C. Bart, E. Tilevich, S. Hall, T. Allevato, and C. A. Shaffer. Transforming introductory computer science projects via real-time web data. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 289–294, New York, NY, USA, 2014. ACM.

[6] A. C. Bart, E. Tilevich, S. Hall, T. Allevato, and C. A. Shaffer. Transforming introductory computer science projects via real-time web data. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 289–294, New York, NY, USA, 2014. ACM.

[7] M. Ben-Ari. Situated learning in computer science education. *Computer Science Education*, 14(2):85–100, 2004.

[8] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational researcher*, 18(1):32–42, 1989.

[9] M. Buckley, J. Nordlinger, and D. Subramanian. Socially relevant computing. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 347–351, New York, NY, USA, 2008. ACM.

[10] A. Bundy. Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1(2):67–69, 2007.

[11] J. Carter, D. Bouvier, R. Cardell-Oliver, M. Hamilton, S. Kurkovsky, S. Markham, O. W. McClung, R. McDermott, C. Riedesel, J. Shi, and S. White. Motivating all our students? In *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education - Working Group Reports*, ITiCSE-WGR '11, pages 1–18, New York, NY, USA, 2011. ACM.

[12] L. N. Cassel. Interdisciplinary computing is the answer: Now, what was the question? *ACM Inroads*, 2(1):4–6, 2011.

[13] J.-I. Choi and M. Hannafin. Situated cognition and learning environments: Roles, structures, and implications for design. *Educational Technology Research and Development*, 43(2):53–69, 1995.

[14] W. Commons. Data science process, 2014.

[15] C. Day. Computational thinking is becoming one of the three R's. *Computing in Science and Engineering*, 13(1):88–88, 2011.

[16] P. J. Denning. The profession of IT: Beyond computational thinking. *Communications of the ACM*, 52(6):28–30, 2009.

[17] P. J. Denning and A. McGettrick. Recentering computer science. *Commun. ACM*, 48(11):15–19, #nov# 2005.

[18] P. DePasquale. Exploiting on-line data sources as the basis of programming projects. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pages 283–287, New York, NY, USA, 2006. ACM.

[19] B. DiSalvo and A. Bruckman. From interests to values. *Commun. ACM*, 54(8):27–29, Aug. 2011.

[20] L. Douglas. The importance of big data: A definition. *Gartner (June 2012)*, 2012.

[21] C. Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer, 2011.

[22] A. E. Egger. Engaging students in earthquakes via real-time data and decisions. *Science*, 336(6089):1654–1655, 2012.

[23] A. Erkan, T. Pfaff, J. Hamilton, and M. Rogers. Sustainability themed problem solving in data structures and algorithms. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 9–14, New York, NY, USA, 2012. ACM.

[24] I. S. for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA). Operational definition of computational thinking, 2011.

[25] A. Forte and M. Guzdial. Computers for communication, not calculation: Media as a motivation and context for learning. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4 - Volume 4*, HICSS '04, pages 40096.1–, Washington, DC, USA, 2004. IEEE Computer Society.

[26] 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. In *Proceedings of the Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups*, ITiCSE-WGR '12, pages 16–38, New York, NY, USA, 2012. ACM.

[27] 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):58–79, #mar# 2013.

[28] M. Guzdial and A. E. Tew. Imagineering inauthentic legitimate peripheral participation: An instructional design approach for motivating computing education. In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, pages 51–58, New York, NY, USA, 2006. ACM.

[29] O. A. Hall-Holt and K. R. Sanft. Statistics-infused introduction to computer science. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 138–143, New York, NY, USA, 2015. ACM.

[30] S. F. Harp and R. E. Mayer. How seductive details do their damage: A theory of cognitive interest in science learning. *Journal of Educational Psychology*, 90(3):414, 1998.

[31] D. Hemmendinger. A plea for modesty. *ACM Inroads*, 1(2):4–7, 2010.

[32] N. Iarocci and G. Amica. Eve. `https://github.com/nicolaiarocci/eve`, 2012.

[33] G. Inc. Exploring computational thinking, 2011.

[34] J. A. Jacobs. Interdisciplinary hype, November 2009.

[35] B. D. Jones. Motivating students to engage in learning: The MUSIC model of academic motivation. *International Journal of Teaching and Learning in Higher Education*, 21(2):272–285, 2009.

[36] B. D. Jones and G. Skaggs. *Validation of the MUSIC Model of Academic Motivation Inventory: A measure of students' motivation in college courses.* Research presented at the International Conference on Motivation 2012, 2012.

[37] J. S. Kay. Contextualized approaches to introductory computer science: The key to making computer science relevant or simply bait and switch? In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 177–182, New York, NY, USA, 2011. ACM.

[38] J. Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, Apr. 2007.

[39] J. Lave and E. Wenger. *Situated learning: Legitimate peripheral participation.* Cambridge university press, 1991.

[40] L. Layman, L. Williams, and K. Slaten. Note to self: Make assignments meaningful. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 459–463, New York, NY, USA, 2007. ACM.

[41] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, 2011.

[42] J. J. McGinley and B. D. Jones. A brief instructional intervention to increase students motivation on the first day of class. *Teaching of Psychology*, 41(2):158–162, 2014.

[43] M. Mitchell, J. Sheard, and S. Markham. Student motivation and positive impressions of computing subjects. In *Proceedings of the Australasian Conference on Computing Education*, ACSE '00, pages 189–194, New York, NY, USA, 2000. ACM.

[44] S. Mneimneh. Fibonacci in the curriculum: Not just a bad recurrence. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 253–258, New York, NY, USA, 2015. ACM.

[45] O. of the Senior Vice President and Provost. Academic implementation strategy for a plan for a new horizon: Envisioning virginia tech 2013-2018. Technical report, Virginia Tech, 2013.

[46] O. L. Oliveira, A. M. Monteiro, and N. T. Roman. From concrete to abstract?: Problem domain in the learning of introductory programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 173–177, New York, NY, USA, 2011. ACM.

[47] A.-C. J. T. F. on Computing Curricula. Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press, December 2013.

[48] S. Papert. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, 1(1):95–123, 1996.

[49] J. Petraglia. The real world on a short leash: The (mis) application of constructivism to the design of educational technology. *Educational Technology Research and Development*, 46(3):53–65, 1998.

[50] J. Pfeiffer. *The thinking machine*. Philadephia, PA: Lippincott, 1962.

[51] C. Rader, D. Hakkarinen, B. M. Moskal, and K. Hellman. Exploring the appeal of socially relevant computing: Are students interested in socially relevant problems? In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 423–428, New York, NY, USA, 2011. ACM.

[52] B. E. Rogoff and J. E. Lave. *Everyday cognition: Its development in social context*. Harvard University Press, 1984.

[53] Y. N. Silva, S. W. Dietrich, J. M. Reed, and L. M. Tsosie. Integrating big data into the computing curricula. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 139–144, New York, NY, USA, 2014. ACM.

[54] D. E. Stevenson and P. J. Wagner. Developing real-world programming assignments for cs1. *SIGCSE Bull.*, 38(3):158–162, June 2006.

[55] J. A. Stone and E. M. Madigan. The impact of providing project choices in cs1. *SIGCSE Bull.*, 40(2):65–68, June 2008.

[56] D. G. Sullivan. A data-centric introduction to computer science for non-majors. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 71–76, New York, NY, USA, 2013. ACM.

[57] T. Thompson and R. Barnes. The engine of successful education reform: Effective teachers and principals. *Commission on No Child Left Behind. The Aspen Institute. Denver,(Oct. 2009)*, 2009.

[58] L. Torgo. *Data Mining with R, learning with case studies*. Chapman and Hall/CRC, 2010.

[59] M. Waldman. Keeping it real: utilizing NYC open data in an introduction to database systems course. *J. Comput. Sci. Coll.*, 28(6):156–161, #jun# 2013.

[60] A. Weinberg. Computational thinking: An investigation of the existing scholarship and research. In *School of Education*. Fort Collins, Colorado State University, 2013.

[61] Z. Weiner. How introductory physics problems are written. `http://www.smbc-comics.com/comics/20131202.png`, Dec 2013.

[62] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[63] S. Yarosh and M. Guzdial. Narrating data structures: The role of context in cs2. *Journal on Educational Resources in Computing (JERIC)*, 7(4):6, 2008.

[64] Z. Zografski. Innovating introductory computer science courses: Approaches and comparisons. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, pages 478–483, New York, NY, USA, 2007. ACM.