HipHopathy, A Socio-Curricular Study of Introductory Computer Science

by

Omoju Adesola Miller

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Science and Mathematics Education

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alice A. Agogino, Chair
Professor Bernard R. Gifford
Teaching Professor Dan D. Garcia

Fall 2015

HipHopathy, A Socio-Curricular Study of Introductory Computer Science

Omoju Adesola Miller

Abstract

HipHopathy, A Socio-Curricular Study of Introductory Computer Science

by

Omoju Adesola Miller

Doctor of Philosophy in Science and Mathematics Education

University of California, Berkeley

Professor Alice A. Agogino, Chair

This work presents a study investigating social and curricular factors that lead to student retention and attrition in introductory Computer Science at UC Berkeley. CS10 and CS61A are identified as critical gateway classes in the curricula pipeline. To address the impedance match between the performance of CS10 students in CS61A, a new curricula Data Science unit, with a culturally-relevant learning experience, has been created for CS10. Preliminary results show agreement with theoretical predictions and significant improvement over previous efforts. The central argument of this work is the belief that historically underrepresented students in Computer Science increase their affinity for and ability to persist in Computer Science classes when they feel like what they are learning has some relevance outside the Computer Science classroom. This work introduces a learning framework for the computational exploration of data, within the context of an introductory Computer Science class. The work presented here has profound implications for future studies of how culturally resonant curriculum may one day help solve the problem of low-representation of female and ethnic minority students in the field of Computer Science.

To Oluwaroju

Contents

List of Figures

List of Tables

Acknowledgments

I want to acknowledge my colleagues at Berkeley who pushed me to realize my true voice. Alice Agogino, my stalwart adviser, who believed in me from the very beginning. Who stood by me as I dipped and dabbled in a lot of things historically outside of my academic domain. Her presence served as my beacon of light, an image that gave me a picture of who I could be.

I want to thank Bernard Gifford who challenged me to bring my authentic self to my research. Who didn't hesitate to hold back both praise and critique. Without whom, the hiphopathy approach would not have existed.

I want to thank Dan Garcia, who made himself available to me, and showed me what a master teacher is. Who let me find my intellectual home in the BJC community. Whose academic presence is the living embodiment of the "Beauty, and Joy" of computing. Whose true delight in teaching as become the solid benchmark that we all measure ourselves against.

I want to thank Brian Harvey for his fastidiousness and unrelenting devotion to an ethical and moral computing. Whose understanding of the social implications of computing is light years ahead of the field. I want to thank John DeNero, Nate Titterton, Michael Ball, Samir Makhani, and Glenn Sugden for the endless conversations they listened to, as I shaped my understanding of computation and learning.

I want to thank the faithfulness and support of my friends, Feyisayo Ojomo, Ugochi Umelo, Delali Attiope, Carla and Tony Wicks; friends who stepped in the gap and allowed me to excel in my academic journey as I parented my child in tandem.

To the amazing ladies that became my UC Berkeley crew-the two live writing crew, "don't stop, get it, get it," Dr Celeste Chavis , Dr Tierra Bills, Dr Krystal Young and soon to be Dr Tia Madkins. The incomparable Dr Jay Dub herself, Jennifer Wade, who spent uncountable hours with me, helping me wrestle my ideas down to form. Who constantly made sure that I stayed true to my intentions of using my craft as a tool for social justice. Through whom I learnt what it was to aspire to greatest in the face of overwhelming odds. To my BGESS crew, Ryan Shelby, David Moody, and others with whom I shared many a evening enjoying "chocolate night."

To my family, my son Oluwaroju, his father Rodric, my parents and my siblings, thank you for being there; each of you had a role to play in helping me become the woman that I am today. To my professional community, CODE2040, my supporters at Kapor Center for social justice, to Nicole Sanchez, Chris Busselles, and Sheila Humphreys, thank you for allowing me to waltz into your lives and creating a space where I was safe to experiment and learn.

Finally to the multitude of Black girls who have been led to believe that computer science is not for them, this work is for you. I want you to know you that not only are you wanted, that you belong, and your voice is needed to usher in the revolution. Keep slaying!

Chapter 1

Introduction

1.1 The Challenge

One fateful day 1952, Stanford Law graduated 112 students, of which 5 were women. One of these five would later rise to become the first female Supreme Court Justice of the United States. Sandra Day O'Connor, graduated third in her Stanford law class, excited about her future in law, only to have her hopes dashed as she could not secure a job! Only one position opened itself to her, that of a *Legal Secretary* (Harper, 2012). It was so rare for women to graduate from law school that society hadn't caught up to that fact. And yet, today, law has overcome these hurdles.

One of most pressing questions for my generation to answer is that of equalizing participation in our economic life, and particularly in the new technical economy. In my field of computer science (CS), there is a near absence of women and underrepresented ethnic minorities (African Americans, Hispanics, Native Americans, Pacific Islanders and persons with disabilities) (Trauth et al., 2012; Gürer and Camp, 2002; Zweben, 2010; NSF, 2013). This statement should be alarming to anyone, because it means that nearly 70%—women, ethnic minorities and persons with disabilities—of the American workforce is not fully engaged in the technology sector. The civil rights leader, the Reverend Jesse Jackson, has in recent years postulated that this lack of gender and ethnic representation is the new frontline of the civil rights movement (Hardy, 2014; Jackson, 2014). Whether one agrees with that rhetoric or not, what is conclusive is that equalizing participation in technology can no longer remain a background issue.

From the outside looking in, one may be easily swayed to believe that what computer science suffers from is the twin evils of sexism and racism. However, a closer examination into the numbers reveals something far more complex. CS suffers from the compounding effect of four separate challenges:

1. A lack of presence of CS in K-12 education.

2. The under-production of post-secondary degrees in CS.
3. The underrepresentation of women in CS.
4. The underrepresentation of ethnic minorities in CS.

For high school students, historically there have been two classes in which CS content are taught. These classes are AP Computer Science A (APCS A) and AP Computer Science AB (APCS AB). The main difference between the two classes are as follows: in APCS A, students are introduced to object-oriented design and problem solving, while in APCS AB, in addition to what is covered in APCS A, students are also introduced to the formal analysis of algorithms and advanced data structures. APCS AB is the equivalent of a full year of computer science at the college level (Astrachan et al., 2009).

In August 2008, the College Board–the body that designs and administers standardized test and curricula for K-12 education–announced the elimination of the APCS AB examination, leaving behind APCS A. This announcement would prove to be the necessary precipitate that would catalyze the conversation around the decline of computer science at secondary and post-secondary institutions, since its heyday in the 1980s.

In 2011, around 20,000 students took the APCS A exam, of which 19% were female. From figure 1.1, you can see in comparison, over 300,000 students took the AP calculus exam, of which 47% were female—figure 1.2. To make these figures more concrete, this means that in 2011, fewer than **3%** of the almost one million high school students taking AP Science, Technology, Engineering, and Mathematics (STEM) exams participated in APCS A.

Longitudinal data collected by the College Board reveals that APCS AB exam participation had consistently hovered around 20,000 since 1997. Over the long run, the College Board determined that it was no longer financially feasible for it to administer the exam, and it was eliminated for reasons of severe under-enrollment. This low participation in APCS A and APCS AB is important because research has shown that students who participate in these examinations are eight times more likely to pick CS as their major (Mattern et al., 2011).

One of the major challenges with equalizing participation in computer science is that CS has not generally been one of the de facto high school classes. What makes this fact important is that if a student is not related to someone who already works in the technology field, there is a high chance that the student will not know technology exists as a potential career path. Unlike other professions like law, medicine, or even certain kinds of engineering, computer science as a field and university major is relatively new. If one is to use the founding of professional organizations as a rough proxy for the formal recognition of fields, we can see that CS is a relative new comer. The American Medical Association (AMA), the premiere learned society for medicine was founded in 1847, 168 years ago. The American Bar Association (ABA) was founded in 1878, 137 years ago. In comparison, the Association of Computing Machinery (ACM) was founded in 1947 (Alt, 1972), while the Computing Research Association (CRA) was founded just in 1972!

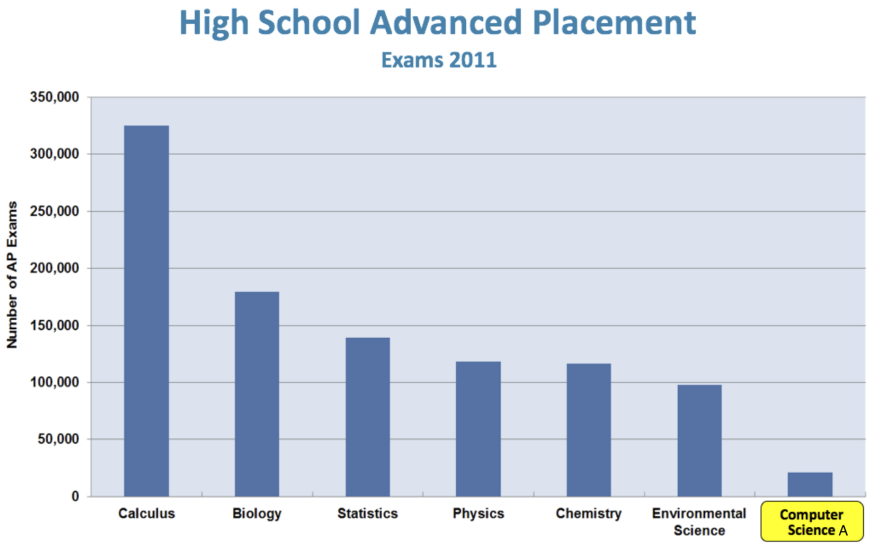


Figure 1.1: High School Advanced Placement Exams, 2011. (ACM, 2012)

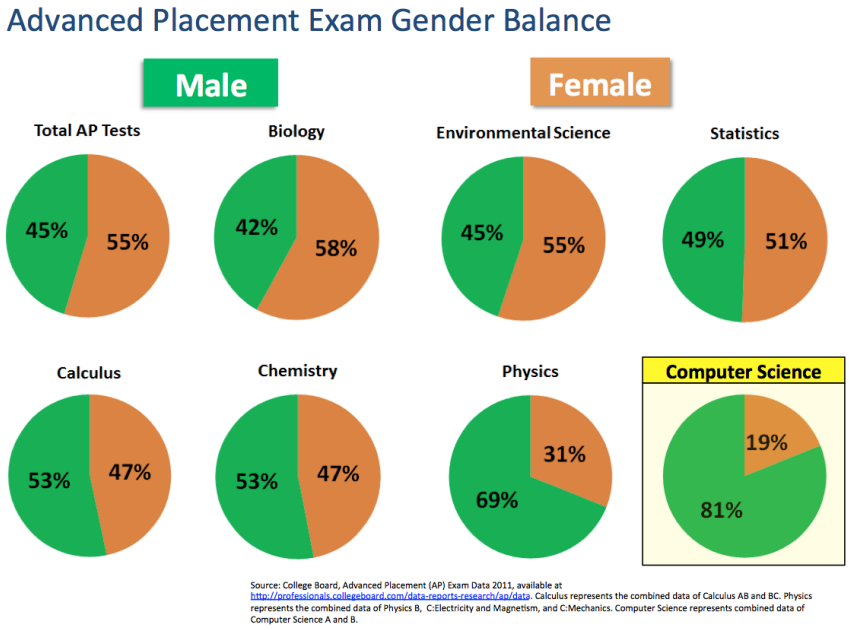


Figure 1.2: High School Advanced Placement Exams by Gender, 2011. (ACM, 2012)

Not withstanding the low participation of computer science in K-12 environments, one could argue that the same goes for law, medicine and engineering. Most high schools do not have an "Introduction to Law" class, neither do they have an "Intro to Medicine." Yet students go on to university and seek those majors out. So the lack of presence in K-12 cannot be the full story.

If one holds true to the maxim that *you need to see it, to be it*, then perhaps we can look to media to fill this gap for early computer science exposure. However, unlike medicine or law, which has a plethora of mainstream media shows, the same thing cannot be said of computer science, unfortunately. This fact is important because research has shown that the media plays a role in helping young people discover potential career paths. To quote Kitzinger, *et al.*,

> "Role model theory suggests that representations of women in SET[1] may be important in showing young people that women can develop successful careers in science, engineering and technology. Experimental and survey research shows that the media 'exert a demonstrable impact on children's occupational knowledge and role identification' and that 'previous experience with (or information about) a successful woman in a traditional male occupation decreases gender bias in evaluation and selection decisions made by both student and professional judges. "' (Kitzinger et al., 2008)

This lack of positive computer science primers from the media further isolates youth from exposure to computer science as a desirable field. With this knowledge, the challenges facing equalizing participation has to be expanded to include:

5. A lack of positive CS role models in the media.

From figure 1.3, we can see that in the 1980s, more than 30% of all graduating universities CS majors were women. By 2010, that number was less than 20%. There are two folk theories relating to women and CS. The first says women are not good at maths, while the second says that maths is a strong indicator of CS aptitude. These two folk theories are tied together to explain the low participation rates of women in CS. Nevertheless, from 1.4, we see that intended CS majors don't have the strongest maths SAT scores (Kinnunen and Malmi, 2006). From data, we see that this lack of *alleged* maths aptitude alone can't explain the low participation rates of women in CS.

This issue of underrepresentation in computer science is not only pertinent to computer science education, but is also of direct importance to the health of the national economy. According to data from the U.S. Bureau of Labor Statistics, for every 150,000 computing

[1]Science, Engineering and Technology (SET)

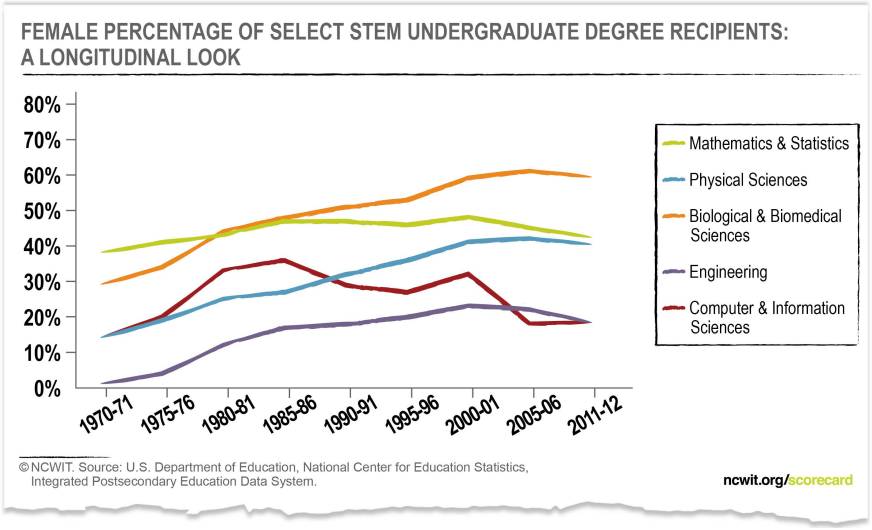


Figure 1.3: Female Percentage of Select STEM Undergraduate Degree Recipients

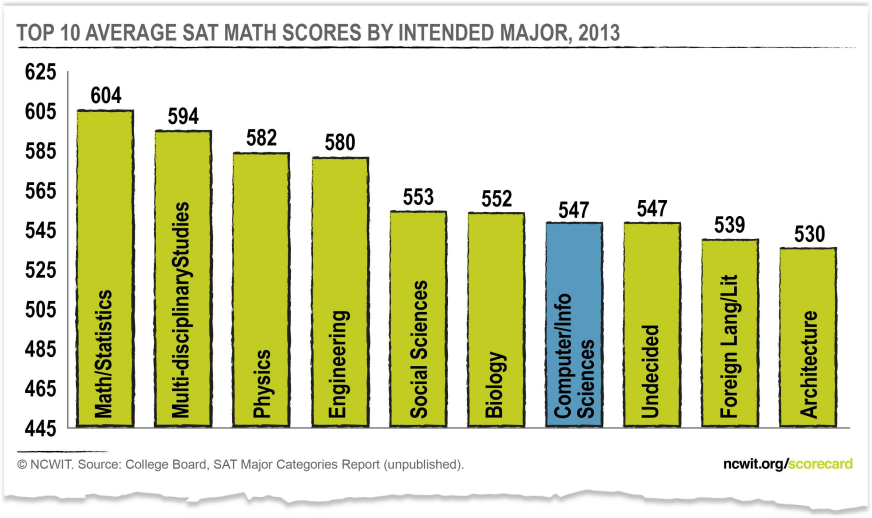


Figure 1.4: Top 10 Average SAT Mathematics Scores by Intended Major

jobs available, as a nation, we produce roughly 55,000 computing graduates to fill those jobs (BLS, 2012). Furthermore, the Bureau has projections which show that 51% of the 9.2 million STEM (Science, Technology, Engineering and Mathematics) jobs between the years 2010 to 2020 will be in computing. Since 70% of the population is underrepresented in computing, this makes the issue of equalizing participation in computer science education a national issue.

Figure 1.5 shows the ethnic breakdown of the population along gender lines. From Figure 1.6, we can see that as of 2010, less than 20% of computer science degrees are obtained by women; to achieve parity this should be 50%. Similarly, less than 20% of computer science degrees are obtained by underrepresented minorities; to achieve parity this should be 28%. Underrepresented minority men are almost at parity in the attainment of undergraduate CS degrees as of 2010.

Based on recent civil rights agitation in addressing this racial and gender gap in technology, some of the major technology companies have moved to transparency as the first step towards righting this balance. While 20% of all CS degrees are obtained by women, in the workforce of the major tech companies, their numbers are not commensurate. Using Google Inc., as an example at the time of this writing, its female technical workforce is represented at 18%, while its ethnic minority technical workforce was at 3% (Google, 2015). Most of the other technology companies have a similar demographic distribution with regard to race and gender.

To bring this issue home, particularly with respect with Google, Inc. In 2010, they conducted a survey on a sample of their US based employees (ACM, 2012). They found the following results:

- Nearly all CS majors (98%) reported being exposed to CS prior to college, compared to less than half of non-CS majors (45%). The nature of the exposure varied from reading about CS, after-school programs or camps, to middle or high school CS classes.
- Those who went on to major in CS were more likely than non-majors to have had a CS class offered in their high school.
- CS majors were more likely to have known that CS was a possible career path when they were in high school.

What makes these finding even more difficult to swallow in light of the numbers, is that according to the ACM Education Policy Committee[2]:

> "Computing jobs are among the fastest growing areas of employment in the United States ... These occupations pay extremely well, providing opportunities

[2]http://www.acm.org/public-policy/education-policy-committee

> for U.S. workers to embark on dynamic careers, enjoy a good standard of living, and contribute to the innovation that drives the country's economic growth."

Furthermore, computing jobs can create an attainable entrée into the middle class for many Americans, particularly ethnic minority persons who face disproportionate challenges with regards to economic upward mobility.

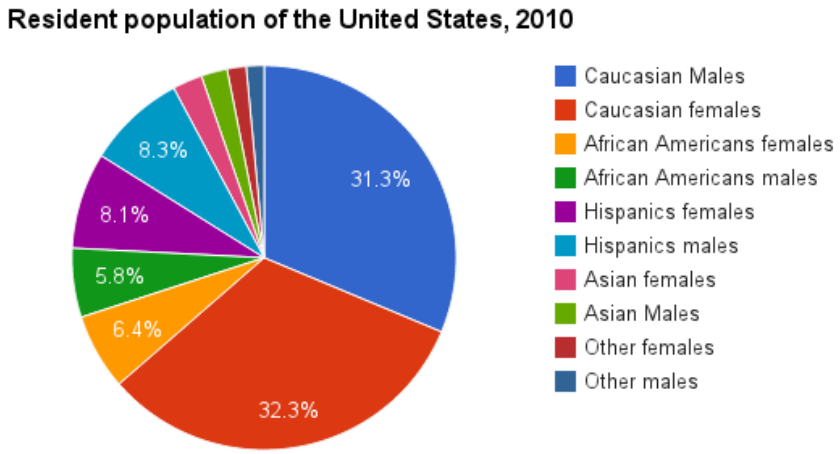


Figure 1.5: United States Census by Gender and Race, 2010

The issue of equalizing participation in a field is not something new. The law profession was wrestling with the same challenges that CS is presently facing in the 1970s. There were also similar calls for law to up their diversity efforts. As early as 1963, a mere 4% of all law students were women. By 2012, law school enrollments, along gender, had reached parity. With regards to ethnic minorities, in the 1980s, they constituted 8% of law students, but by 2012, ethnic minority students constituted 24% of the graduates (Harper, 2012). Certainly if a field like law—who like computer science is not available at the high school level—can overcome these hurdles, then surely we can do the same.

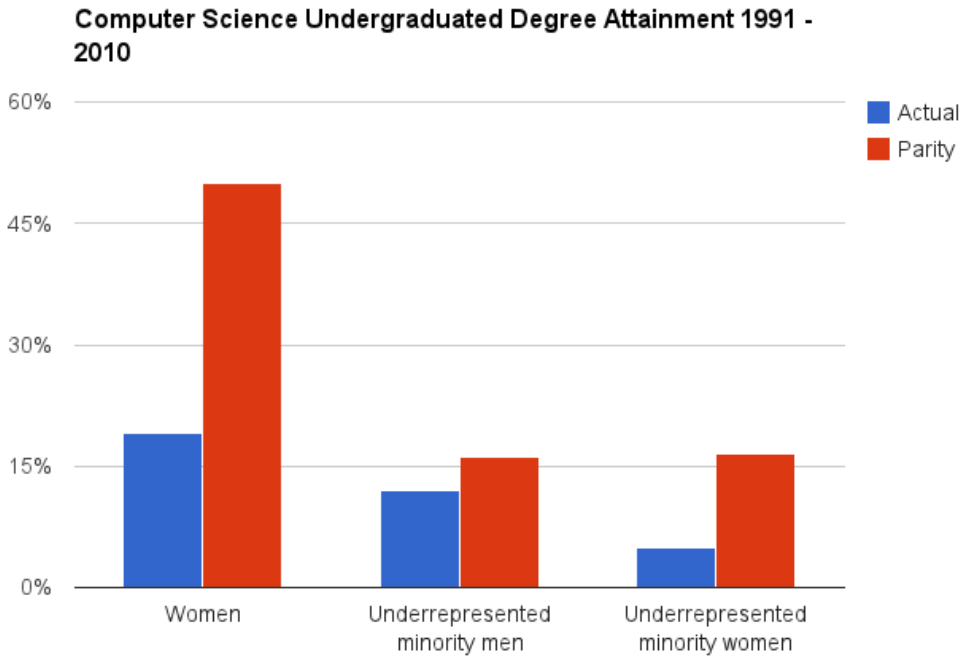


Figure 1.6: Undergraduate CS Degree Earned, 1991-2010

1.2 Proposed Solution

As a means to address the low numbers of computer science instruction at the high school level as well as the under-production of computer science degrees at the collegiate level, a new AP CS Principles course has been developed to broaden participation in computing. Instead of focusing mostly on programming and algorithms, this new curriculum

> "introduces students to the central ideas of computing and computer science, to instill ideas and practices of computational thinking, and to have students engage in activities that show how computing and computer science change the world." (College Board, 2015)

UC Berkeley has been chosen as one of the pilot sites for this course (Astrachan et al., 2011a). The course with the title "Beauty and Joy of Computing (BJC)," is implemented in UC Berkeley's "exploratory" series as CS10—a CS course designed for non-majors.

Inspecting the demographics of the university, as can be seen in figure 1.7, we see that the student population is mostly White and Asian; as a result, this location is limited in its ability to serve as a testing ground in assessing the efficacy of the new AP CS Principles

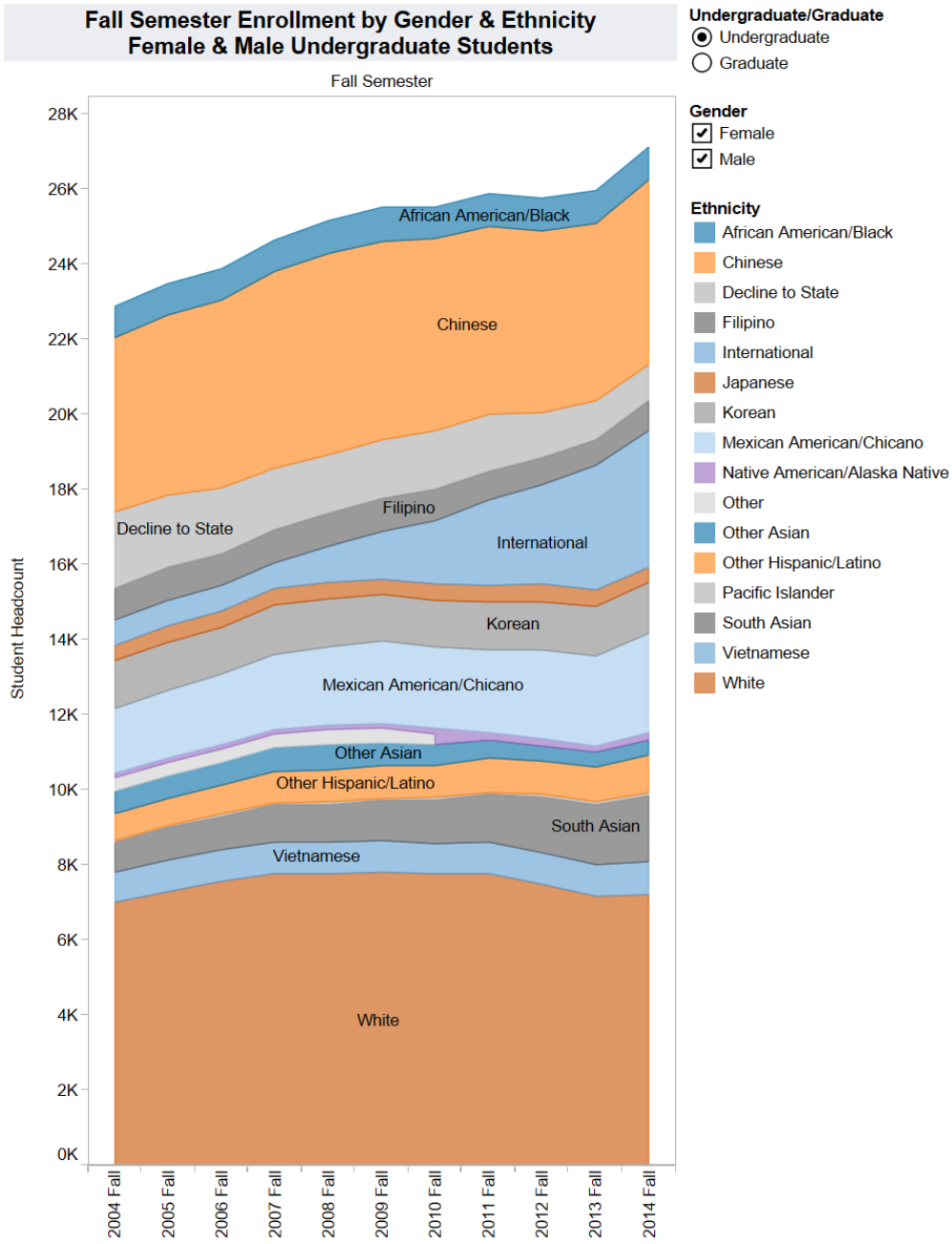


Figure 1.7: Ethnicity Distribution of Undergraduates at UC Berkeley, Fall 2004 - Fall 2014. (UC Berkeley Office of Planning & Analysis, 2015)

curriculum in attracting ethnic minority students. However, the university provides an excellent environment where we can investigate the impact of the BJC curriculum on female students.

CS10 uses uses a visually rich programming environment called Snap*!* an extension of MIT Media Lab's Scratch (Resnick et al., 2009), developed to accommodate first-class procedures and first-class lists. In the Snap*!* environment, students build their scripts by clicking together lego-like block structures.

Snap*!* has several pedagogical benefits, including eliminating the notoriously frustrating experience of learning the syntax of most text-based programming environments. Snap*!* also has a feature that renders it unable to create syntax errors, adhering to the low-floor, high-ceiling model of pedagogically sound introduction to CS learning tools (Grover and Pea, 2013).

1.3 Theoretical Rationale

The rationale for carrying out this research is two-fold. First, this research analyzes the impact of CS10 on the computer science experiences of underrepresented students (mostly female) at UC Berkeley.

CS10 was specifically designed for non-majors to broaden participation. Since its inception, the course has been phenomenally successful in attracting female students, with the spring 2013 offering breaking the 50% female barrier[3]. One of the goals of this work is to see if CS10 is able to serve as a successful gateway into the major, and more important, if a significant portion of the female students feel a strong sense of CS belonging.

Assuming the class fulfills its mission, the second reason for undertaking this study is to address the impedance match between the performance of CS10 students as they progress to CS61A—Structure and Interpretation of Computer Program, the first class designed for majors. The working hypothesis is that students who have previously taken CS10 have a challenging time transitioning to the next course, CS61A because they do not realize that they already have "computational thinking," the most essential skill necessary to succeed in the class.

Moreover, students are struggling with learning to program in Python, the programming language in which computation is realized in CS61A. From anecdotal feedback, we have observed students getting bogged down with understanding syntax. Because CS10 students have heretofore been programming in a *typo* free[4] environment—Snap*!*—they spend more

[3] It is important to note, this is the first computer science course, since UC Berkeley has been tracking student course data, that has ever achieved that feat.

[4]Unlike other text-based languages like Python and Java, Snap*!* doesn't allow free-form typing, as a result, syntax errors are eliminated. Because of this capability, some have erroneously regarded visual programming languages as "syntax *free*" environments. In fact, these visual languages do have syntax,

cognitive resources mediating between what they already know, which is solving problems with a computational approach realized in a visual language, and trying to map this new syntax driven text-based language unto that.

Where as, their counterparts who have not had a previous programming experience before CS61A seem to be doing better in the class, than the students who have had prior programming exposure in a visual blocks based environment, which poses a conundrum. Often the academic community works from the assumption that prior programming experience is beneficial to success in student performance in the first class designed for majors. What we have experienced at UC Berkeley does not support this hypothesis, and this study investigates why this is so.

Based on the working hypothesis that it is the transition from Snap*!* to Python that is responsible for the impedance match, a new curricular unit called *Besides Blocks* has been designed. Besides Blocks is a curricular unit that transitions students from Snap*!* to Python by situating the two languages within a higher computational thinking context. The pinnacle of the Besides Blocks unit is the Python data unit.

1.4 Theoretical Framework

This work draws from the cognitive perspective on learning, emphasizing the understanding of concepts, reasoning, and how that influences problem solving (Greeno et al., 1996), with particular emphasis on constructionism (Papert, 1980a). A lot of the ideas for this research are motivated by the ideas that Seymour Papert touted in his seminal book *Mindstorms: Children, Computers, and Powerful Ideas*. Papert in the 1980s created the LOGO programming environment to demonstrate how children's computational thinking skills can be harnessed through the use of a computational object to think with, in his case, the "LOGO turtle."

With LOGO, children could create figures on a screen by entering a few basic commands to the computer. These commands took the form of naturally occurring English words like "forward," accompanied by a number that determined how many steps the artificial turtle of the LOGO environment would move. Through these commands, children were able to learn programming, but more important to gain an intuitive understanding of geometry.

The keywords in the LOGO programming language were words that children were already familiar with. This act gave each student a non-threatening way to engage with programming, thereby creating a low barrier entry into introductory computer science. The Snap*!* programming language is an intellectual design descendant of LOGO via Scratch.

Scratch was designed specifically to improve on the aspects of LOGO that researchers believed led to its eventual decline in education. In order to realize Papert's dream of computational fluency for all, researchers at the MIT Media lab extended the LOGO programming

because a programmer writing code in the medium still has to arrange the blocks in the permissible order.

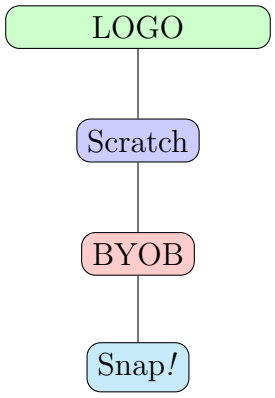


Table 1.1: From LOGO to Snap*!*

language to make it more "tinkerable," i.e., they moved the medium of writing instructions from text to visual blocks, and they also created an online social network around the language, which allowed for the sharing of ones work with a wider audience (Resnick, 2012).

To achieve this, they took away LOGO's ability to create procedures. This change in the underlying design of Scratch led the developer Jens Mönig to create a version of Scratch which allowed for the creation of procedures, allowed for the use of functions as data, and the creation of arbitrary list. This version was called BYOB for Build Your Own Blocks. Later on, the name was changed to Snap*!*.

The most influential aspect of Papert's work is the idea that knowledge is built upon already existing ideas and schema that students have. The way we learn, Papert states is either by building upon our pre-existing theories, or creating new concepts, and then integrating them into our broader theories. Papert's idea that education should have cultural resonance is an idea that is fundamental to the creation of *all* good curricula. In his book, *Mindstorms*, Papert lays out how his love for gears was a great entry point and an intuitive cultural tool that he could use to understand formal systems.

> "In the Foreword of this book I described how gears helped mathematical ideas to enter my life. Several qualities contributed to their effectiveness. First they were a part of my natural "landscape," embedded in the culture around me. ... Second, gears were part of the world of adults around me and through them I could relate to these people. Third, I could use my body to think about the gears. ... And finally, because, in a very real sense, the relationship between gears contains a great deal of mathematical information, I could use gears to think about formal systems." (Papert, 1980a, pp. 11)

These ideas that Papert lays down in this excerpt should be understood as the foundational articles upon which every culturally relevant pedagogical work should be built. Appropriating the turtle in the LOGO programming environment gave children a way to think about the principles of computation and the practice of programming.

This study integrates both the cognitive and socio-cultural views of learning. Often in academic research, cognitively oriented studies in learning are not brought to bear with socio-cultural studies in understanding cognition (Sperber and Hirschfeld, 1999). This is especially detrimental when we are studying learning and cognition within groups whose culture varies from the norm[5]. Because some of the research questions of this work is interested in understanding the role that cultures plays in cognition, it synthesizes scholarly research in both the cognitive and socio-cultural traditions.

Socio-Cultural Perspective

There is a wide body of scholarly research, both empirical and theoretical, highlighting the educational challenges of ethnic minority students. However, most of the research is often presented through a deficit framework (Lee, 2008); a cultural orientation that says it is the students that are broken and not the system in which they are taught. Some scholars are beginning to recognize that students of color community and culture are often devalued by the public education system. The response has been some research that looks at how we can draw upon cultural resources to improve classroom instruction, e.g., "funds of knowledge" (Moll et al., 1992) and cultural modeling (Lee, 1995). There has actually been an extensive tradition of culturally-responsive pedagogy in mathematics education. The work of Tara Yosso, which uses community's cultural-wealth has been developed in response to the deficit model of ethnic minority students education in the United States (Yosso, 2005).

Computational thinking can give us the opportunity to bring in culturally-relevant pedagogy into CS. Because of its recent ascendancy, there is very little research both empirical and theoretical on how culturally-responsive pedagogy that supports computational thinking can be done. It is notably that similar work is been done in computer science with the emergence of the ethno-computing field (Eglash et al., 2006; Eglash et al., 2013; Magerko et al., 2013). Emerging scholars are starting to engage scholarship in the process of how we draw upon communities' cultural resources within CS.

Unfortunately, when one encounters the words "**culture**," or "**socio-cultural**," it triggers that part of our mind that associates with ethnic minorities. The word *culture* in this context often foreshadows "othering." But culture encompasses all. It remains a challenge to bring about good culturally-responsive teaching in multi-cultural environments, especially in a place like UC Berkeley.

[5]The current norm in CS is White and Asian male.

1.5 Research Context

As a computer scientist whose life experience sits squarely at the intersection of both avenues of underrepresentation in the field, i.e., gender and ethnicity, issues of inclusion were starting to become harder and harder for me to ignore. It was for this reason I decided to head back to graduate school and investigate why students who looked like me were not fully represented in computer science. I was fortunate enough to have decided to investigate this issue at the same time it was getting addressed by society writ large.

In the last few years since I have started this line of inquiry, I have been fortunate enough to have ample exposure to various learning environments where I could observe and see how students experienced computer science education both in-school and after-school settings. Spring 2011, I had the opportunity to teach middle school and high school girls *Introduction to App Development* through Iridescent Learning's Technovation Challenge, an after-school program designed to introduce girls to new product development in 8 weeks, with the support of female industry professionals. Technovation Challenge gave me hands on experience in broadening participation.

The Technovation Challenge used a drag-and-drop programming environment *AppInventor*[6], designed by Hal Abelson—author of *Structure and Interpretation of Computer Programs*; the definitive textbook that has been used by many leading institutions and is considered one of the best attempts to understanding computation.

As great as the program was, and as easy as AppInventor was to use, it came up quickly against a tough ceiling. The apps that were created were simple prototypes that needed to be re-engineered in Java to be of real use. Once again, educators needed to progress from a drag-and-drop environment to a text-based environment. It seemed that once the interest for computer science was ignited in the student, we as the computer science education community still needed to create something that will continue to fan those flames and not let them die out in the wilderness that the AP CS Java class had become.

While my exposure to Technovation cheered me on to believe we could turn the ship around, it left me posing this question, would the program had been just as successful in igniting the curiosity for computer science if it had been unisex? Furthermore, as an educator and parent, I felt uneasy with the unsaid notion that *boys* were the problem, that it was the boys in introductory computer science classes that were responsible for chasing the girls away. Boys who might have more experience with computer science, who in their enthusiasm to be in a place where their interest in the subject is celebrated, leads them to sharing more and more, often unaware that they are dominating the classroom conversation. A behavior that has the unintended consequence of exclusion, by leaving novices feeling out of their depths; novices who often are girls.

[6] AppInventor is based on Android mobile operating system designed by Google. AppInventor provides a graphical interface to abstract away Java and XML that Android uses.

This realization led me to accept an opportunity to design and teach a unisex middle school computer science summer program at Stanford University. While I did witness some of the stereotypical "bad boys" behavior, I realized something deeper still. I realized that every time we create an all-girls program as our attempt to design a "safe learning environment" for girls, we rob boys of the opportunity to socialize well with girls in the computer science classroom. More important we fail to correct deleterious behavior that stems from gendered notions of computer science belonging. This experience made me realize single-sex environments might not be the panacea that we had hoped for, and furthermore, single-sex environments were just a way of postponing the inevitable.

In the Stanford middle school computer science summer program, I experimented with building transitional curriculum designed to help students transition from a graphical programming language to a text-based programming language. These and other experiences are what led me to conduct a formal analysis on the effectiveness of CS10 in attracting historically underrepresented students to computer science.

1.6 Research Framing

The central question that drives my research is, what are the socio-curricular factors that lead historically underrepresented students to choose computer science? Is there something about computer science culture that makes it difficult for students who are not White or Asian males to choose this field? Like most scholars in the field, I started my investigative thinking about this issue by focusing on the missing parties, the women and men who are not represented. As I continued on my investigative journey, however towards the end of my investigation, I realized that I had been wrongly thinking about the issue. I had inadvertently adopted a deficit framework, instead of being agnostic in my approach, investigating both the represented and the underrepresented.

The work that is presented in this research stems from these lines of inquiry and leads to unexpected conclusions that make computer science as a field, as well as its culture all the more endearing. It is its strong tight knit cultural valence that is also its greatest weakness and its largest opportunity.

Chapter 2: Related Work. Chapter 2 explores the conceptual ideas that ground this research, namely: computational thinking, the barriers to learning and the complex nature of underrepresentation in computer science education.

Chapter 3: Research Methods Chapter 3 presents formative and mixed-method research conducted to understand the social and curricular cues that surround the decision to major in computer science at the undergraduate level. Specifically this study seeks to answer the following research questions: R1) What are the factors that contribute to the underrepresentation of historically marginalized groups in computer science? R2) What impact

does CS10 have in attracting female students into the major? R3) What role does socio-culturally-responsive curriculum play in attracting underrepresented students into computer science? and finally R4) How does a graphical-based programming language affect students sense of CS identity?

Chapter 4: Design of a CS0 Course for Inclusion Chapter 4 presents an overview on the design and creation of the Beauty and Joy of Computing curriculum. The approach is juxtapose against that of UC Berkeley's CS61A, the first course designed for majors. Through this exploration, we see that as class sizes grow exponentially, some of the approaches that were successful in attracting historically underrepresented students in CS become challenging to apply. Approaches that allow students to design their own projects, that allows for culturally-relevant curriculum, and incentivizes creativity instead of grade optimization.

Chapter 5: Design of Python Data Lab Chapter 5 presents an in-depth analysis of the design and creation of a culturally-responsive data module as part of the laboratory supplements to the Beauty and Joy of computing. It discusses the challenge that comes with bringing in a scientific inquiry approach to knowledge into introductory computer science as well as the use of contemporary datasets as a means of injecting education into culture.

Chapter 6: Findings and Discussion Chapter 6, presents findings from the formative and mixed-methods research are presented along the following dimensions: Mentorship, Family, Outsider Experience, Belonging, Programming, Efficacy, CS Efficacy, Social Implications of Computing, Gendered Ideas of Intelligence and on Besides Blocks, with particular emphasis on the Data Module.

Chapter 7: Conclusion and Future Work Finally, Chapter 7 presents findings from the formative and mixed-methods research are tied to related work to help answer R1) What are the factors that contribute to the underrepresentation of historically marginalized groups in computer science? The findings from the data analysis investigating: attitudes towards CS, identity and self-efficacy as well as belief about CS ability are analyzed to help answer R2) What impact does CS10 have in attracting female students into the major? Both quantitative and qualitative studies help answer R3) What role does socio-cultural responsive curriculum play in attracting underrepresented students into computer science? and finally, R4) How does a graphical-based programming language affect students sense of CS identity. In addition, conclusion and recommendations are presented, paying special emphasis to moving the conversation from *broadening participation* to that of inclusion, and the need to bolster up alternative post-collegiate paths into computer science.

Chapter 2

Related Work

"It is very much like learning a new language, the people who started off young have such an advantage, they are bilingual in a way, so, I am like the older person, trying to use Rosetta Stone."

— A female UC Berkeley CS major.

To adequately design a sustainable solution to the issue of *equal*zing participation in the technology sector, we first have to go back and take a look at our past. First let's unpack the term. Equalizing participation means to have everyone represented in some proportion commensurate to their representation in the population.

This begs the question of how we got to the point where everyone was not adequately represented. Another way of expressing the concept behind "equalizing participation" is to say "how do we fix the challenges that have resulted from gender and racial discrimination?" Until we engage the heart, mind, and will in courageous conversations around these issues, we will not be able to see our way out of this quagmire.

To gain a solid sense of the issues in this space, and examples of approaches that have been tried to equalize participation, three different lines of inquiry will be examined:

1. The nature of innovation in computer science education.
2. The barriers to learning.
3. The misconceptions pertaining to computer science.

One can opine that our focus as learning scientists should be the transfer of deep knowledge, helping our students attain a thorough understanding of the subject at hand. The journey to get there often starts with the experience of the novice learner, and it is for this reason that this study's inquiry is situated in the "introductory to computer science" space. To quote Sawyer et al.,

> "Learning is not just about content. It needs to focus on becoming . . . helping people to grow in capabilities and awareness and disposition which includes learning content" (Sawyer et al., 2006).

This becoming that Sawyer et al. speaks of is at the core of solving our CS challenge. For it is in *this becoming* that it seems certain men, women and/or ethnic minorities have difficulty. In addition, the challenge of transforming a non CS affiliated person into a CS person, now occurs at a very interesting and innovative time in the history of computers in education.

2.1 Computational Thinking

Computer science education is undergoing a tremendous change, thanks in part to a convergence of fortuitous innovations and trends. In the fall of 2011, Stanford University professors Sebastian Thrun and Peter Norvig announced a new initiative. They announced that they would be bringing their Artificial Intelligence (A.I.) class to the world through a massive open online course (MOOC); it was dubbed "The Stanford Education Experiment" by the press. In the first offering of that class, over 100,000 students signed up (Leckart, 2012). The phenomenal number of people who signed up for the course concretized in the minds of many a great desire for CS outside the walls of the academy. A year prior to the Stanford A.I. class experiment, a start-up, Codecademy, built a web application that allowed anyone to learn programming via carefully scaffolded curriculum with the addition of in-browser programming environments. Swiftly after that came the computer science track on the Khan Academy site and the computer science education technology train, has not stopped rolling since.

Roughly around the same time as these online education innovations, a similar innovative spark was lit in "Introductory Computer Science" circles through the introduction of an idea, "Computational Thinking (CT) for all." As explained in the previous chapter, sometime in 2008, College Board pulled its APCS AB examination from its roster. This incident happened during the tenure of Dr Jeannette Wing as the Assistant Director for Computer and Information Science and Engineering (CISE) at the NSF. She had previously started the conversation around the need for a different kind of introductory computer science curricula; through the injection of an idea she termed *computational thinking*. With the canceling of the exam, spurred on by the assistant director, NSF led the charge for the creation of a new AP CS class that would, in theory, be capable of equalizing participation was the idea they coalesced around Wing's computational thinking (CT) (Wing, 2008).

What exactly is computational thinking and why did many believe it would be able to accomplish that which the old approach had failed to accomplish? In short, CT can be understood as the act of learning to think like a computer scientists, placing an emphasis on the cognitive processes that computer scientist use in solving problems. According to Wing,

> Computational thinking is taking an approach to solving problems, designing systems and understanding human behavior that draws on concepts fundamental to computing." (Wing, 2006)

The focus is on understanding the many layers of abstraction that define computer science. What it is not is *computer programming*. CT is not merely learning how to program. While programming might be used in exploring the many layers of abstraction that is computer science, CT does not require the use of a machine. In the Wing's view, the CT processes are capable of being carried out by any computational substrate, be it a human, a digital computer, or whatever the future substrate that computation would be realized in.

This emphasis away from *programming* to computational problem solving appealed to many computer science educators. In addition, CT's focus on conceptualizing and not programming, on ideas not artifacts, and most important, its idea of centering computational problem solving around how humans think. Moving the emphasis away from the machine to the human came to be seen as the right approach. Computer science educators quickly joined in the conversation and began agitating for broad education reform in K-12 computer science curricula. In aggregate, their thinking went thus: perhaps CT—if implemented successfully—might do that which AP CS AB realized in Java, and previously in C++, was not able to accomplish, i.e., engage a broad spectrum of students and not just the mostly male "dungeons and dragons" playing, socially-maladjusted, obsessive compulsive types, or those who started early as children. (Harvey, 2014)

Scholars have suggested that CT be included into traditional core curriculum which all students are mandated to learn. These scholars have suggested that CT should be taught along side reading, writing, and arithmetic. This view asserts that a broad overview of computing is general knowledge that is a fundamental component of being an educated person in the 21st century (Isbell et al., 2009). As such, there is a desire to have CT introduced across the K-12 school curriculum (Barr and Stephenson, 2011). There remain many issues with regards to CT yet to be finalized, issues like what form does it take at each level of schooling, how is it defined, how do we develop curriculum that supports it, and finally, how do we build assessments to determine when one has mastered CT?(Aho, 2011; Denning, 2009; Linn et al., 2010; Brennan and Resnick, 2012)

Exploring Computational Thinking

As with most aspects of CS, the term computational thinking is very broad. A cursory search for the term on the Google search engine yields curriculum that is targeted to elementary school students all the way to college level and beyond. At present, the term is so broad that it quickly becomes meaningless. Aho, has suggested using the term with a well defined model of computation, whose semantics are clear and matches the problem being investigated (Aho, 2011). While Aho's edict would clarify the term, as well as ground it in rigorous CS, evidence suggests that this approach hasn't found much favor in the CS education community.

Denning, on the other hand, argues that computational thinking isn't actually unique to CS, but rather is a notion that grows out of the nature of *science* itself (Denning, 2009). He sees the whole CT movement as a fad that the CS education community is clinging to, as a hope for reversing the declining enrollments in the field[1]. If other sciences at their core, deal with the relationship between multiple layers of abstraction, and their coordination in solving problems in an algorithmic way, then computation thinking will in fact be *old wine, in new wine skin*. Instead of computational thinking as a way to define CS, Denning posits his "Great Principles Framework," as an alternate way of defining CS. He separates the principles that define the field from the practices that are utilized. His framework is outlined in table 2.1.

The Great Principles Framework

Core Principles	Core Practices
Computation	Programming
Communication	Engineering of Systems
Coordination	Modeling
Recollection	Applying Computational Thinking
Automation	
Evaluation	
Design	

Table 2.1: The Great Principles Framework for Defining CS (Denning, 2009).

Grover and Pea argue that computational thinking is distinct from other kinds of reasoning, that it is in fact something unique to CS (Grover and Pea, 2013). The linchpin of their argument is the role that *abstraction* plays. In their view, abstraction is seen as the cornerstone of computational thinking. They go on and explicitly list the ideas that define computational thinking, as elaborated in table 2.2.

On the other hand, the AP CS Principles curriculum framework takes a page out of Denning's book, separating CT practices from its CT concepts (College Board, 2014). For the AP CS Principles CT curriculum, its fulcrum rests on **creativity**. The committee places emphasis on the need for students to apply creative processes when developing their artifacts. Table 2.3 outlines the CT practices and concepts that define computational thinking in the AP CS principles framework.

How to build an assessment for computational thinking has been understudied, or under formalized. A lot of the papers in this space don't often touch on this topic. However, Brennan

[1]At the time Denning published his article, CS enrollments were in decline. Since then, the enrollments have started increasing for certain demographics.

The Core Elements Defining Computational Thinking

Abstractions and pattern generalizations (including models and simulations)
Systematic processing of information
Symbol systems and representations
Algorithmic notions of flow of control
Structured problem decomposition (modularizing)
Iterative, recursive, and parallel thinking
Conditional logic
Efficiency and performance constraints
Debugging and systematic error detection

Table 2.2: Elements Defining Computational Thinking (Grover and Pea, 2013).

AP CS Principles Curriculum Framework

CT Concepts	CT Practices
Big Idea 1: Creativity	P1: Connecting Computing
Big Idea 2: Abstraction	P2: Creating Computational Artifacts
Big Idea 3: Data and Information	P3: Abstracting
Big Idea 4: Algorithms	P4: Analyzing Problems and Artifacts
Big Idea 5: Programming	P5: Communication
Big Idea 6: The Internet	P6: Collaboration
Big Idea 7: Global Impact	

Table 2.3: AP Computer Science Principles Curriculum Framework (College Board, 2014).

and Resnick make an attempt at formulating CT assessment (Brennan and Resnick, 2012). They have come up with a triangular approach that comprises automated project portfolio analysis, artifact based interviews and design scenarios, based on the blocks programming language Scratch and its online community.

Project portfolio analysis: The research team used a piece of software that mapped each block used in a project into a visualization. This served as a form of formative assessment; allowing the investigator to see the evolution of CT competency as defined by using certain blocks over time.

Artifact Based Interviews: This method helped them find the weakness in their blocks analysis approach. In interviews, it became apparent that despite the *Scratchers* apparent fluency, there were significant conceptual gaps in their computational knowledge. Not surprisingly, students were just copying pieces of code and didn't really know how

they worked.

The Design Scenarios: In a series of three interviews, students were presented with the design scenarios, which were framed as projects that were created by another young Scratcher. The students were then asked to select one of the projects from each set, and:

1. Explain what the selected project does.
2. Describe how the project could be extended.
3. Fix a bug in the project.
4. Remix the project by adding a feature.

Of the three approaches they experimented with, the design scenarios assessment model stands outs. While it is time consuming, and would probably not scale for the conventional classroom model of education, this model would do quite well as a alternate model for software engineering technical interview process. One should note that Brennan and Resnick do not address the issue of CT transfer from the classroom to real-life scenarios, a gap that should be filled inorder to have a robust assessment of computational thinking.

Computational Thinking: a 1980s Redux

"CT" idea is not altogether new. CT as an idea, at its best, is just a rehashing of the same common sense ideas that Alan Perlis and Seymour Papert had advocated for in the 1980s (Grover and Pea, 2013). That being said, CT is certainly an idea whose time has come.

Seeing all the effort that is going into CT and its realization in AP CS principles, one can't help but think, "but we have done this before, and nothing changed." The same fervor surrounded the Papertesque LOGO disciples. These disciples were supposed to usher in a revolution in computer science education. Seymour gave us a vision of what computers could *really* mean in education. It was supposed to be a ***revolution!***. And yet nothing happened. School was able to kill that which was birthed when children, computers and—as we the Papertian devotees say—powerful ideas come together (Papert, 1997; Harvey, 2014; Watters, 2013).

School as realized through the bureaucratic obstacle course that American public education has degenerated to, successfully adopted the idea of learning with computers, but turned it into something that no longer resembled anything that could potentially empower and liberate students in their learning. Instead of learning how to explore complex computational ideas via programming, students were ushered into sterile computer labs and "taught" how to use modern software programs like word processors.

Schools were able to get away with an intellectual perversion of transforming the student from the *creator* of software into the *consumer* of software. The lucky few who came from environments where the parents had more agency than the school system, which often correlates with high-income neighborhoods that are predominantly White, were the ones that

had the luxury of building their computational thinking skills in their *true* CS classes and not the joke that the *technology* classes had degenerated into.

The 1980s computers in education experiment further led to the magnification of the already wide divide between access to computer science education in most American public schools. What made this divide even more difficult to bridge was the conflation of ***access to computer use*** as ***access to computer science education***.

What happened in the 1980s is a clear example of Toyama's amplification theory, the theory states that technology is only an amplifier of human intent. As a result, it can have both negative and positive impacts (Toyama, 2011a). When computers were introduced into schools that had pre-existing human capacity in teacher and parent networks, it found a sure footing and was used as a tool to introduce computer science. In the case where computers were introduced to schools that were already stretched thin in terms of their human capacity both in competent teachers and parent networks, it turned into "typing" and learning to use "Microsoft Word." Jane Margolis has done a wonderful job in chronicling this experience in her book, *Stuck in the Shallow End: Education, Race, and Computing* (Margolis, 2008).

Concomitant with the opportunities and challenges that CT may pose is an examination of the socio-cultural understanding of learning and barriers which might inhibit or detract students from achieving a positive identification with education at large. The next section of this chapter highlights the academic dis-identification theory of learning; which has been shown to particularly hinder the participation of women and ethnic minority students.

2.2 Understanding Barriers to Learning

The late Nigerian American anthropologist, John Ogbu, was one of the foremost scholars in studying the differences in academic performances of majority and minority students, the so called *achievement gap*. At the start of his research, Ogbu tried to understand why this pattern was persistent in many countries like the United Kingdom, the United States and India. He theorized that the gap existed because the minority groups in those countries were a part of a *castelike* social system (Ogbu and Simons, 1998).

Later he noticed that immigrant minorities tend to do well in school, while non-immigrant minorities tend to have the achievement gap. It is this aspect of his work that led him to his *cultural-ecological* theory of African Americans and academic disengagement. He explains that,

> "The theory has two major parts ... One part is about the way the minorities are treated or mistreated in education in terms of educational policies, pedagogy, and returns for their investment or school credentials. ... The second part is about the way the minorities perceive and respond to schooling as a consequence of their treatment." (Ogbu and Simons, 1998)

The most important contribution of Ogbu's work was to explain that the sometimes dysfunctional aspects of African-American cultural orientation towards education in the aggregate ***were*** adaptations, and were **NOT** inherent cultural traits. Before Ogbu's theory, some scholars had believed that the negative association that some African American youth often exhibit with relation to schooling was an inherent cultural trait of the race. Based on this idea, researchers and educators that were interested in creating avenues of engagement for African American youth, seemed doomed in their cause. Important to note in Ogbu's theory was the role of the new Black immigrants, the so called *voluntary* immigrants and their relationship to schooling. Foley 2004 comments that:

> "For the optimistic voluntary immigrant, America is a land of opportunity compared with the harsh economic realities of their homeland. Given their 'dual frame of reference', they do not develop a pessimistic oppositional youth culture that equates school achievement with cultural assimilation and loss." (Foley, 2004)

Perhaps this is the reason why a significant portion of Black students in CS and CS related fields in tertiary institutions and in industry are African immigrants. Later, we shall see that this *cultural* assimilation is what the nascent field of ethno-computing is trying to overcome by exposing computational thinking ideas as embedded within cultural practice and cultural artifacts.

Stereotype Threat

Even with Ogbu's theory, there were still some questions that were left unanswered. For example, there were students who have in some shape or form overcome the negative association with schooling that their peers had; these minority students had come to embrace intellectual achievement as part of their identity. And yet, these students were under-preforming with respect to their majority peers.

Before we delve deeper into this issue, lets take a step back and examine an interesting phenomenon that started occurring in elite institutions among minority students.

During the heady days of the late 1960s, Harvard University, perhaps caught up in the fervor of civil rights movement, decided to start intentionally recruiting African Americans into its freshman class. The best and brightest students were recruited and everyone assumed they would go on and prove to the world how smart they were. The hope was that these students would go on to disgrace the adherents of segregation and myths of biological determinism that posited that Africans and African Americans were intellectually genetically inferior to whites—the eugenists.

Instead of soaring with flying colors, something else happened to these students. In aggregate, they seemed to be underperforming with respect to their peers at Harvard. Some believed

that the students were not up to the task, that perhaps after all, the pseudoscience of eugenics was indeed right. From Harvard, to the University of Michigan, this phenomenon continued to occur. Hoping to unravel this particular knot, the psychologist Claude Steele started his investigations into the issue.

Building on the the work of Ogbu, Steele discovers what he has termed *stereotype threat* theory, to explain the under-performance of seemingly high achieving students in certain environments (Steele, 1997). In his seminal paper, *A Threat in the Air: How Stereotypes Shape Intellectual Identity and Performance* he elaborates on this idea:

> "It is the social-psychological threat that arises when one is in a situation or doing something for which a negative stereotype about one's group applies. This predicament threatens one with being negatively stereotyped, with being judged or treated stereotypically, or with the prospect of conforming to the stereotype. Called stereotype threat, it is a situational threat—a threat in the air—that, in general form, can affect the members of any group about whom a negative stereotype exists . . . for members of these groups who are identified with domains in which these stereotypes apply, the threat of these stereotypes can be sharply felt and, in several ways, hampers their achievement." (Steele, 1997)

The unfortunate aspect of stereotype threat is that it specifically affects students who are strongly identified with a domain, for which a negative stereotype of their group exists. Reminded of the stereotype that African Americans are not good at intellectually demanding fields, those students go on and under-perform.

The negative stereotype about a social group can be triggered by subtle environmental cues. For example, Cheryan and colleagues have found that something as seemingly innocuous as *Star Wars* posters in computer science classrooms are enough to trigger the negative stereotype of women's underrepresentation in the field as a result of their alleged cognitive inferiority with mathematics and abstract ideas (Cheryan et al., 2009). Stereotype threat has been used to explain the underrepresentation of women and ethnic minorities in CS, women's low participation in math, and even the lack of White participation in the hip-hop music genre.

For those students suffering from the condition, if the threat conditions continue to persist, those students might respond by redefining their self-concept to dis-identify with academic achievement in the field. This might explain why some minority youth by by middle school and sometimes high school start to disengage with learning. The appropriate way to understand this disengagement is as a self-preservation response triggered by the perception of a hostile environment (Steele and Aronson, 1995). The same theory can be used to explain why by sophomore year, many female students often opt out of the CS major (Kinnunen and Malmi, 2006). These female students, surrounded by social and environmental cues, constantly reminding them that they do not belong in the field, eventually succumb to this threat and distance themselves from the major.

In the next section, I shall speak about the different responses that have been offered to counter the effects of stereotype threat. I will explore both curricular responses as well as structural and social responses to the phenomenon.

The Treisman Approach

> *"We did not question that minority students could excel. We just wanted to know what kind of setting we would need to provide so they could."*
>
> — Uri Treisman

The good news is that there are some interventions that have proved successful in mitigating the effects of stereotype threat. Contrary to what one might think, if students are under-performing because of threat conditions, giving challenging work to students over remediation has shown to actually reduce this effect. By challenging the students, it shows the students that their teachers and professors believe in them, and believe in their intellectual ability.

In the 1970s, the mathematician Uri Triesman set out to create environments that would be conducive to positive intellectual performance of ethnic minority students (Treisman, 1992). In his research he found out that the major reason why ethnic minority students were under-performing at UC Berkeley was because they were studying alone! Unlike their Asian counterparts, they didn't have study groups that allowed them to learn from each other and gain a reasonable estimation of where they stood in the class.

Based on the findings of Triesman's research, an anti-remedial program was constructed to attract students who saw themselves as well prepared. It was opened to all students of all races. The program emphasized group learning and a community life focused on shared interest in mathematics. This created an environment that was both intellectually challenging and emotionally supportive. The results of their program was that "Black students with Maths SAT scores in the low-600s were performing comparably to White and Asian students whose Maths SATs were in the mid-700s" (Treisman, 1992).

Probably influenced by Triesman's results, Steele did a similar experiment at the University of Michigan that also proved the efficacy of intentional, racially integrated, learning and living spaces that reduced the effect. What Steele's research team did in particular was to allocate a cross-section of incoming freshmen to a particular dormitory, all the while ensuring that there were significant numbers of ethnic minority students there. They then went on to offer enrichment classes to all the students, and allowed students to opt in and out at will. Instead of saying something along the lines of:

> "Dear Minority student: Congratulations on your admission to the University of Michigan. The University of Michigan is a difficult institution. You are going to

> need a lot of help and we are here to help you,"

their approach rather was to say something like this:

> "Listen everyone, we know that you guys are stellar students, however adjusting to college life can be difficult, as a result, we are offering a series of classes to help you get better with writing, maths and so on."

The ethnic minority students who were struggling were not singled out. Instead, no reference was made to their position as ethnic minorities, they were just a part of a freshman dormitory for students. When this approach was taken instead of an ethnic minority remediation program, grades for Black students in the inclusive program went up, as well as their retention. As opposed to the ethnic minority only remediation programs that already existed on the campus (Steele, 1997). It also important to note that this experiment didn't hurt White students grades at all; they did just as well, but for the Black students, their grades went up.

Ethno-Computing: A Curricular Response to Stereotype Threat

In the academy, stereotype threat has been widely accepted as an adequate explanation for the low participation rates of women and ethnic minority students in CS. To correct the effects that the theory highlighted, scholars who are particularly interested in equalizing the participation of ethnic minority students have taken a curricular approach that seeks to inject education into culture. These scholars position computational thinking ideas as part and parcel of ethnic minority culture and not an external intellectual notion foisted upon them from the majority culture that they equate with cultural loss. This approach has roughly been regarded as *ethno-computing*.

The goal of this kind of research is to expose implicit mathematical/computational sophistication that is embedded in cultural artifacts. The hope is that this will help fight the dis-identification with education that as be seen in some African American students. Foremost in this line of inquiry is the work of Ron Eglash (Eglash et al., 2006; Eglash, 2007; Eglash et al., 2011; Eglash et al., 2013).

Fractals and African Designs

In the 1980s, Eglash happened to notice that if you look at aerial photographs of some African villages, you will see arrangements that look like geometric fractals. Armed with a Fulbright scholarship, Eglash spent a year traveling on the African continent looking for villages that were assembled in a self similar pattern, harkening back to fractal geometry. Through his research, he came to understand that these designs didn't just occur by accident. In fact the Africans had done them intentionally and understood ideas that the West deemed mathematically complex.

Through his inquiry, he came upon the village of Logone-Birni, Cameroon. From figure 2.1, you can see that the village itself is arranged in a fractal pattern. He went on to discover that the royal insignia for the village is actually of fractal design, a rectangle within a rectangle within a rectangle and so on, like the one in figure 2.1. Further, he realizes that the path through the palace uses a recursive path. As one goes through the path, ones has to get more and more polite as the path gets closer and closer to the throne room. He realizes that the architects that built the palace consciously mapped that societies social scaling onto geometric scaling (Eglash, 2007).

Whenever these kinds of research are conducted that investigate the intersection of culture and cognition, many questions quickly arise. For example, questions like "Isn't it just intuition, and not mathematics?"

Eglash's strongest indication that the use of these mathematical ideas are conscious come from his exploration of the mathematical ideas underlying African divination systems.

Separate from Eglash, I had also suspected that mathematical ideas were consciously embedded in African culture by my experience of growing up Yòrubá in Lagos, Nigeria. My grand father was a Yòrubá king, and as such—even though my family practiced Christianity like many africans, they still held on to their indigenous beliefs and still practiced some of the religious rituals—I grew up exposed to the knowledge that there lay other forms of knowing that were completely African and alien to western cognitive traditions.

Eglash discovered that the African divination system—Ìfá in Yòrubá—which is done by drawing lines on a powdery surface or through the use of shells, uses what will be recognized in the West as a pseudo-random number generator using deterministic chaos to generate a number; a series of binary sequences through addition in modulo 2, just like in the parity bit check of a computer. In this excerpt from his 2007 TED talk, Eglash traces the genealogy of the computer from African divination systems. He elaborates as follows:

> "And the most interesting thing I found out about it was historical. In the 12th century, Hugo of Santalla brought it from Islamic mystics into Spain. And there it entered into the alchemy community as geomancy: divination through the earth . . . Leibniz, the German mathematician, talked about geomancy in his dissertation called 'De Combinatoria.' And he said, 'Well, instead of using one stroke and two strokes, let's use a one and a zero, and we can count by powers of two.' . . . George Boole took Leibniz's binary code and created Boolean algebra, and John von Neumann took Boolean algebra and created the digital computer." (Eglash, 2007)

Based on his findings, he designed simulations that supported cultural design activities as a way of bringing in ethnic minority culture into the math and computing classroom. When the ethno-computing curriculum was adopted, the research team saw a statistically significant

Figure 2.1: Aerial image of Logone Birni, Cameroon, (Eglash, 1999).

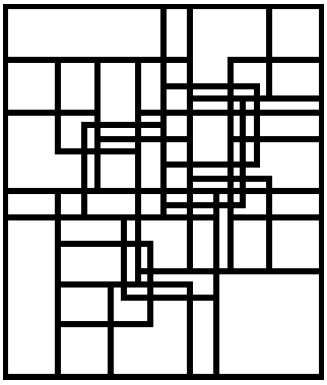


Figure 2.2: Fractal model of the palace constructed by Eglash, (Eglash, 1999).

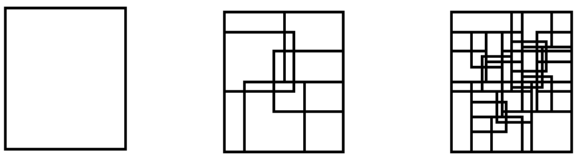


Figure 2.3: First three iterations of the fractal model, (Eglash, 1999).

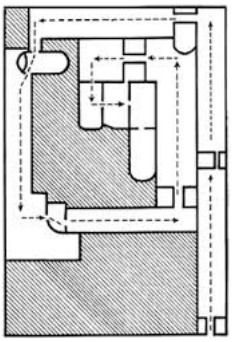


Figure 2.4: Path to the throne room in the palace, (Eglash, 1999).

increase in the scores of ethnic minority students in pre-collegiate math and computing (Eglash et al., 2013).

Apart from Eglash, other scholars are also adopting his approach as a means of equalizing participation in CS. Magerko and his collaborators have built a tool that they call EarSketch. EarSketch is a music computation tool with a Python API that has been successfully incorporated into a CS Principles class. The goal of the tool is to allow students to make hip-hop beats by writing Python code (Magerko et al., 2013).

While this approach is certainly interesting, it is important to note that it is not novel. Similar work has been done in the field of English and literature. A foremost example that comes to mind is the work of Carol Lee and her experience with cultural modeling. She uses the preponderance of reasoning embedded in African American vernacular culture of signifying to help students transfer that knowledge into the domain of canonical literary analysis (Lee, 1993; Lee, 1995).

Exploring Creativity as means of Combating Stereotype Threat

Another approach that some scholars are taking as means of equalizing participation in computing with regards to ethnic minority students is exploring the role that creativity plays in learning. They are taking this approach since research has shown that African Americans in particular have high self-reported aptitude with regards to creativity (Kaufman, 2006). This approach is particularly more interesting now since creativity plays a prominent role in the seven big ideas of computing as defined in the AP CS curriculum framework, and as we can see from table 2.3 (College Board, 2014).

As previous iterated, Carol Lee saw how fluent her African American urban students were with playing the dozen or *signifying* and used that as a scaffold on which to build literary analysis. At the core of the approach is an understanding that creativity can be used as a tool for engaging African American youth. The "*dozen* is a type of verbal dueling in which usually male antagonist address pointed barbs, often in rhymed couplet form at one another"

(Dollard, 1990); sometimes these couplets take the form of 'Yo Mama jokes. Wald elaborates about the cultural role the dozen plays in the life of African Americans in this quote,

> "At its simplest, it is a comic concatenation of "yo' mama" jokes. At its most complex, it is a form of social interaction that reaches back to African ceremonial rituals. Whether considered as vernacular poetry, verbal dueling, a test of street cool, or just a mess of dirty insults, the dozens has been a basic building block of African-American culture."(Wald, 2015)

Along the same vein, Kaufmann in his work with regards to the nature of intelligence, noticed that creativity could be used as a measure of giftedness instead of the standard IQ test. Further, he noticed that based on the studies that they did, creativity appeared to be the one place where there were no significant achievement gaps based on race or gender. If anything, some of the results tended to favor African Americans as having higher creativity in some certain measures (Kaufman, 2006).

In Kaufmann's study, his subjects were asked to rate their ability along the following domains: science, social, visual art, verbal art and sports. They found that African Americans in the study were less likely to fall prone to gender stereotypes. The women rated themselves higher on the science-analytic factor than their majority peers, while the men rated themselves as significantly more creative in the visual-artistic factor than European American, Hispanic American and Asian American men; even though is stereotypically female and was generally rated higher by women than men.

What this implies is that some of the gendered approaches that have been designed as interventions to get more women in CS may not necessarily apply to African American women. Challenges around gender and computing, might be more valid with respect to White and Asian women. This line of reasoning is supported by Riegle-Crumb and King. They found that after accounting for high school preparation, the odds of declaring a physical science or engineering major are two times greater for Black males than for White males, and Black females are closer than White females to closing the gap with White males (Riegle-Crumb and King, 2010).

Because the present norm in CS education is a White and Asian male student, a lot of the work that is done with respect to equalizing participation gets lumped together.

However, while the majority of the United States population is underrepresented in present CS, the approaches that are needed to increase participation is not "one size fits all". A lot of the work that gets done in this space often conflates minority in technology with women in technology. If women in technology is what is put forward, then it often means non-ethnic minority women in technology.

It is very difficult to find a racially balanced and gender balanced environment to study CS education. As a result, a lot of what we find may work for one subgroup and not the other.

Having addressed the issues that pedagogy and schooling might pose in equalizing participation in CS, there are still other goals yet to be solved. Heretofore, we have been operating under the assumption that with all things being equal, women and ethnic minority students will pick CS. That assumption may not be true, mainly because computer science is suffering from a major image problem. In the next section of this chapter, I will outline and discuss the cultural challenges that the field faces in creating healthy computer science environments, i.e., environments that are not predominantly dominated by White and Asian males.

> *"The single story creates stereotypes, and the problem with stereotypes is not that they are untrue, but that they are incomplete. They make one story become the only story."*
>
> — Chimamanda Ngozi Adichie

2.3 Misconceptions Pertaining to Computer Science

Tracy Camp's seminal paper on the shrinking computer science pipeline helped to shed light on the issue (Camp, 1997). In the paper Camp optimistically predicted that the number of female CS degree recipients would increase by 2007. Unfortunately, the opposite has happened. It has been 18 years since her article was published, 18 years of debates around the issue of women and ethnic minority participation in CS, 18 years of ideas, 18 years of interventions and nothing has really changed. Yes, there are some perturbations in the system that seem as if we are finally moving the needle, but in aggregate there has been little improvement.

In another paper, Camp this time in collaboration with Gürer goes on and broadly outline all the challenges that cause women to fall out of the computer science pipeline (Gürer and Camp, 2002). Challenges from their first interactions with computers in pre-school all the way to navigating work-life balance. From the causes she outlined, one of the more pernicious one is the perception of the field itself. I believe this is the most difficult to overcome because it is the one that we can not manipulate as educators and academics. It is one that has to do with society's challenge around gender inclusion. It is the one that rests squarely on the shoulders of *patriarchy*.

The media depiction of the CS sphere broadly falls into two categories. It is either hyper-masculine, mostly White or Asian male environment, with egos running wild, a.k.a. the *brogrammer* culture; otherwise it is still a White or Asian male, but this time, isolated male nerd who codes away on his computer all alone.

This pervasive negative stereotype of practitioners and the societal norms of CS have been shown to aid in dissuading interested parties from the field (Cuny and Aspray, 2000). The

classic image that people have of people who do CS is from the show *South Park*, and in particular of *Jenkins* the "computer guy" also known as the "griefer." It is interesting to note that *Jenkins* only appears in two episodes of *South Park* out of 257, and yet if one googles "South Park Computer Guy," there are over 2.4 million hits! From that, one can draw the conclusion that this stereotype is now embedded in contemporary culture. Quoting from the *South Park* wiki, *Jenkins*

> "...appears to be extremely overweight. This is supported by his excessive consumption of junk food and energy drinks. He is mostly bald with matted, unkept brown hair on the sides of his head. He wears a gray t-shirt with black trim and black sleeves. He has acne on his face, wears round framed glasses, and occasionally will have bits of food crumbs on his stomach. He also wears a wrist brace, most likely due to carpal tunnel syndrome, which would probably be brought on by his hand posture, poor health, poor diet, and lack of exercise" (South Park Archives, 2015).

If the image that students have of people who do CS is *Jenkins*, one can easily see how that makes the field unappealing to many people.

In the study of the experience of undergraduate majors in CS at Carnegie Mellon University, Fisher and Margolis found that the cultural perceptions of CS could be a major obstacle in getting more women interested in the field. They found the gap between the reality and stereotype of the qualities required in a good CS major was wide (Fisher et al., 1997). Furthermore, their study revealed that women experience CS in different ways than men. The female students were drawn to the field because they wanted to apply the knowledge to solve more socially conscious problems like health care and education, which makes the context of CS meaningful for them. Unfortunately, many people hold the misconception that CS is a socially isolating field that does not contribute to the betterment of society (Margolis et al., 2000). This may be one of the numerous reasons why there is a persistent decline in the number of students entering the field.

Writing Women and Ethnic Minorities out of CS History

In recent years, Hollywood as turned its narrative lens to Silicon Valley. Studios have started releasing stories that explore the trope of the lone founder who saves the day; we see these tropes used in films like 2010's *The Social Network* or 2013's *Jobs*. These movies celebrate and valorize the *take-no-prisoners*, *scorched-earth* approaches of the mostly male founders of technology companies. These movies do little to improve the already negative image of the field. In fact one can argue that they do more to damage the image of the field.

The reason why one has to investigate the role of media in equalizing participation in CS is because media is one of the major means by which our stories are told and disseminated. One of the more challenging aspects to creating a more balanced—if not slightly tilted towards

inclusion—representation of the field is twofold. The stories that get told, and the characters that get written out of those stories.

To drive this idea home, lets take a closer analysis of the 2013 movie *Jobs*, celebrating the life of Apple Inc co-founder Steve Jobs. The movie tells the story of his rise to become one of the most celebrated entrepreneurs of the 20th century. According to Megan Smith, chief technology officer for the United States, in an interview with Charlie Rose,

> *"There are these incredible photographs from the launch of the Macintosh in the 80's, and the Rolling Stone pictures that were published. The historic record shows this group of 10 people in a pyramid–actually 11, seven men and four women. Every photograph you see with the Mac team has Joanna Hoffman, who was the product manager, a great teammate of Steve Jobs, and Susan Kare who did all the graphics and user interface on the artist side. None of them made it into the Jobs movie. They're not even cast. And every man in the photographs is in the movie with a speaking role. It's debilitating to our young women to have their history almost erased.[2]"* (Smith, 2015)

You can see the image from the original team in figure 2.5. This image particularly illustrates the the gap between reality and perception in CS, particularly with women. This image is far from the hostile image that we believe of CS. In it are the leaders of the field, frolicking in their office with babies.

In addition to the lack of female representation in their own history in computer science, this issue is even more heartbreaking when there are virtually no stories told of the role that ethnic minority characters played in the history of computing. Incredible stories like that of Katherine Johnson, a Black female NASA mathematician, who is portrayed in figure 2.6. It was Katherine's job to calculate the trajectory for the space flight of Alan Shepard, the first American in space; John Glenn, the first American to orbit earth; and Apollo 11, the first human mission to the moon. (Makers Profile, 2015).

Schulte and Knobelsdorf highlight other misconceptions that exists around CS. They reveal that unaffiliated persons usually conceive computer science as the professional use of computers. Unaffiliated people do not understand what CS is, and they see themselves as outsiders who can never function in the field. The unaffiliated perceive CS as a field that cannot be learned like other subjects, they see it as this mysterious thing that only special men can do (Schulte and Knobelsdorf, 2007).

Many interventions have been implemented to improve the number of women in computing. The National Center for Women and Information Technology (NCWIT) was established to address this issue. There has been a proliferation of mentoring for women in computing, after school programs to under-gird math and science pedagogy and so on.

[2]Correction: The pyramid photo featured eight men and three women (Susan Kare, Rony Sebok, and Patti Kenyon). Joanna Hoffman wasn't in the photo

Figure 2.5: The original Macintosh team in the 1980s

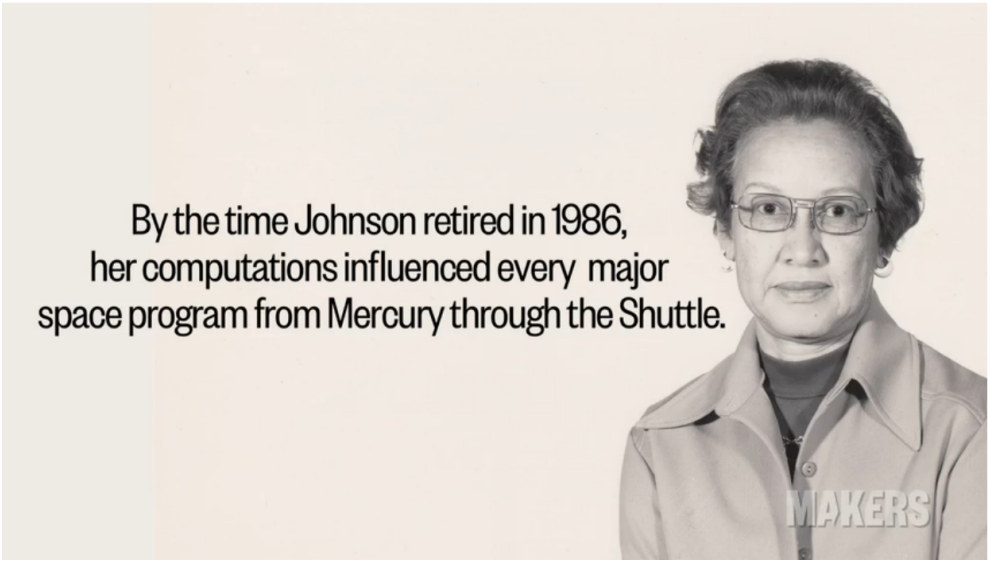


Figure 2.6: Katherine Johnson, NASA Computer Scientist (Makers Profile, 2015).

In spite of the millions of dollars thrown at the issue, the number of women entering the CS pipeline at the undergraduate level still continues to drop. Blum suggest that what we need to do is reframe the conversation (Blum et al., 2007). Blum and colleagues state that focusing on gender emphasizes differences and creates an oppositional framework, which in turn, makes the experience not as fulfilling for the already-outnumber women.

By creating a gendered framework, researchers have implicitly reinforced the misconception of CS as a field that excludes underrepresented racial minorities. A White or Asian only zone is yet another nagging misconception of CS. Blum, *et al.* have proposed a *cultural framework* for improving the diversity in the field. In their estimation, what is needed is a change in the perception of the field and a change in its culture and norms.

Challenging CS History through Media

To combat these misconceptions about CS, giant technology companies like Google Inc, have launched socio-curricular campaigns. They launched "Made With Code," to get girls interested in programming (Google, Inc., 2015).

Code.org—one of the larger non-profit organization in the CS Education space—launched their annual "Hour of Code" campaign in 2013, with the intention to get a million children writing at least one line of code. The campaign proved exceptionally successful, resulting in over 60 million children participating over the next two years (Code.org, 2015).

Recently, the White House has also joined the chorus of organizations involved in attaining gender equity in CS with their "The Untold History of Women in Science and Technology" Campaign (The White House, 2015). Even the President of the United States got involved, and wrote his first line of code in 2014. For that special event, he was coached through the process by a young African American girl. The symbolism of that moment wasn't lost on most people. The first African American President, writing the first presidential line of code, whilst the first lesbian chief technology officer of the United States looks on with the founder of Code.org, an Iranian American; in a room that show cased the computational artifacts of Lieutenant Grace Murray Hopper, the founder of COBOL programming language and one of the few women who plays a prominent role in the history of computing. That particular video of the President went viral on social media.

As of the writing of this dissertation, there are several documentaries that have either been created or are in the works speaking specifically to the dearth of women in CS. To raise awareness of this issue, and to support those doing great work in media to correct the image of women in computing, NCWIT in collaboration with Google has gone as far as to create the first ever Science, Engineering, Technology (SET) Award[3] for "Portrayal of a Female in Technology" (NCWIT, 2014).

[3]The SET Awards are presented to movies, TV series, radio and television news programs, print and online journalism for inspiring, authentic and encouraging entertainment portraying and promoting the fields of science, engineering, technology, and math.

Fortunately, women in CS has become the celeb course du jour, that even non-visual media are playing their part in bringing awareness to this issue. National Public Radio ran a story recently titled When Women Stopped Coding, to help the public gain understanding of the over-representation of White and Asian males in the field (Henn, 2014).

The question we have to ask is "Will all these efforts lead to an increase in female participants in CS in the K-16 pipeline over the next coming years?" Only time will tell.

The Myth of Innate Ability

Another persistent misconception surrounding CS is the myth of *innate ability*, or the idea of *biological determinism*. This particular myth isn't limited only to CS as a justification for the near absence of women and minorities. This myth is often called upon to justify the absence of historically underrepresented groups in all STEM fields.

For example, Lawrence Summers—the then president of Harvard University—suggested that the reason why there was a dearth of female faculty in Science and Engineering was because males have more "intrinsic aptitude" for high-level science (Riley, 2005).

These myths about innate ability serve as a cultural marker for who gets to belong in this field, and can be used for aspiring computer scientist to self-select out. Further, these myths can assuage the conscience of apathetic teachers, professors and administrators to justify the lack of representation of certain groups in their field.

The other side of the myth of innate ability, is the "boy hacker icon" (Margolis et al., 2000), the so-called *bros* that were *born programming*. The common myth that some hold, which simply states that some expertise may come from practice, but at the end of the day, there is a sort of inherent "ability" that enables certain people to become better experts than others.

On the surface this seems like a reasonable view on expertise or giftedness. However, after consulting the research literature on the acquisition of expertise, the evidence points to the contrary. Research findings contend that deliberate practice is the main determinant between expert and accomplished amateur (Ericsson et al., 2007). Unfortunately, many students still believe that a pre-requisite for CS success is previous forays into hacking. These students get even more intimidated when they see how students celebrate those classmates who have been "coding since 6."

Even when one moves away from folk psychology to academic research, the problem of innate ability still seems to rear its head. For example, Bornat and his student, Dehnadi, initially claimed to have conceived of a test that can tell students apart based on their aptitude to succeed in CS, even before the student ever learned to code! They claim:

> "We have found a test for programming aptitude, of which we give details. We can predict success or failure even before students have had any contact with any programming language with very high accuracy, and by testing with the same

> instrument after a few weeks of exposure, with extreme accuracy." (Bornat and Dehnadi, 2006)

The Bornat, *et al.*, 2006 paper was greeted with much fanfare. It was sliced, diced, and parsed by a lot of distinguished software engineers and highly respected computer scientists. Jeff Atwood—Founder of StackOverflow and the blog Coding Horror—wrote about it (Atwood, 2006), Alan Kay—pioneer of object oriented programming, Turning Award winner, the inventor of the dynabook, precursor to the iPad—was asked about it; he thankfully discredited the theory (Secret Geek, 2008). On and on the talk of innate programming aptitude went, until the initial research team had to retract their findings when their so-called programming aptitude test failed, when held up to the rigor of peer reviewed science (Bornat et al., 2008). It turned out that Bornat was having a psychotic episode which ended up hurting computer science education. Here is Borat's retraction of his initial claim in his own words:

> "Though it's embarrassing, I feel it's necessary to explain how and why I came to write "The camel has two humps" and its part-retraction in (Bornat et al., 2008). It's in part a mental health story. In autumn 2005 I became clinically depressed. My physician put me on the then-standard treatment for depression, an SSRI . . . I did a number of very silly things whilst on the SSRI and some more in the immediate aftermath, amongst them writing "The camel has two humps". I'm fairly sure that I believed, at the time, that there were people who couldn't learn to program and that Dehnadi had proved it. Perhaps I wanted to believe it because it would explain why I'd so often failed to teach them." (Retraction Watch, 2014)

What exactly do we know about how students' learn CS and the predictors of success in CS? Researchers in CS education have historically aligned the challenges that novice programmers go through in their learning journey toward expertise as similar to the developmental challenges children go through, as outlined by Jean Piaget's "cognitive stages of development" theory (Piaget, 1964). Linking Piaget's developmental levels to the ability of programmers, has been one of the major ways scholars have tried to discover the illusive predictor of programming ability. Lister has a good exposition of the varied attempts scholars have made to use Piagetian theory to predict programming aptitude with varied success (Lister, 2011).

In recent times, CS scholars are beginning to use a *neo-Piagentian* theory to understand novice learners. The main difference between classical and neo-Piagetian theory "is that people, regardless of their age, are thought to progress through increasingly abstract forms of reasoning as they gain expertise in a specific problem domain" (Lister, 2011). This approach provides a means of getting beyond the innate ability argument as a reason for the high failure rates that have become commonplace in introductory CS courses. Instead, it

provides us with a new theory of learning that might provide insights into effectively moving novice learners along the learning path towards expertise.

From everything we know, we have discovered that learning is a progression through legitimate developmental phases that start with preoperational reasoning, through to concrete reasoning, and finally arriving at the formal reasoning we have come to associate with expert programmers. Even though we cannot prove it absolutely, from research we have seen that their isn't some special *innate* pre-determiner for CS aptitude. All those that have tried to make such a claim have woefully failed at this task. Perhaps because we do not yet have a good working knowledge of how people learn CS, many fall back on myths of innate ability to justify the high attrition rates of historically underrepresented students in CS.

The boy hacker icon, coupled with the myth of innate ability conspire to render students who have not had previous programming experiences to believe they are not smart enough for CS. Marvin Minsky, one of Papert's contemporary and a Turing Award winner, touches on this subject in his Turing acceptance essay. He touched on the fact that many ideas around CS have so many pedagogical affordances that can be leveraged to build CS confidence in the novice learner. On debugging he says

> "The idea of debugging itself, for example, is a very powerful concept—in contrast to the helplessness promoted by our cultural heritage about gifts, talents, and aptitudes. The latter encourages 'I'm not good at this' instead of 'How can I make myself better at it?' " (Minsky, 1970)

What Minsky touched on can now today be recognized as the Dweckian "growth mindset," which positions the learner in a cognitive orientation towards the development of a healthy, progressive attitude towards learning (Dweck and Leggett, 1988).

One of the best aspects of solving problems using computation is wrestling with failure. I believe it this aspect of computer science that make people look to experts in the field with awe. CS is a field that constantly challenges what is possible. Often that challenge is done by pushing the boundaries of what we know, specifically by investigating why what we know fails at the boundaries.

If we are serious about equalizing participation in CS, then we need to completely change our orientation around failure, and embrace it as a necessary developmental step in moving novices through to expertise. We need to teach the science of computing more than the skill of programing. We need a overhaul to the *Introduction to Computer Science* curriculum to embrace cultural-relevant education.

2.4 Still Waiting for Superman. . .

The hope is that "computational thinking for all" is not another highly touted education reform around students and computers. But instead a genuine opportunity to use this

moment to finally create curricular reform that allows for a multiplicity of knowing and allows for the creativity of our multi-cultural heritage to shine through.

In 2010, the celebrated filmmaker Davis Guggenheim decided to turn his camera lens towards the American education system. He called his investigative documentary *Waiting For Superman*, a tongue in cheek allusion to the need for a superhero to come in and save the American education system.

However, if history is anything to judge by, this moment may prove to be yet another event in the long list of failures in the puckered history of computers in education. As iterated at the beginning of this chapter, until we—society—get the heart, mind, and will to change, we don't. An organization is not going to evolve its cultural DNA without the leadership of its founder. Just because an idea is sound and has popular support, the problems with schools are not going to magically vanish; we are all *still* waiting for Superman.

The question that is the hardest to get a clear answer for is "Who wears the founder's hat in the K-12 education space?" Is it the local superintendents? The individual school boards of education? The NSF? College Board? Who really has the power to change the cultural DNA of mainstream American schools? It is that person/organization that will determine whether CT as realized in AP CS Principles will finally broaden participation in computer science—at least at the high school level.

Ensuring that we do not leave this section without coming up with some alternative solutions to broadening participation. The challenges that need to be surmounted in the public education space are so daunting as to render the best intentioned persons powerless. In order to do anything, one would first have to change how schools are funded. This is about the only way to get to an even playing field at least with respect to individual school funding. Then, if through some miracle, we are able to achieve that goal, we still need to centralize curriculum around rigorous CS. We still need to train computer science teachers en mass, with the hope that they won't decamp to higher paying high-tech jobs. If all these can be achieved then we will finally be in a position to start tackling the recruitment of women and ethnic minority students into these classes.

If the solutions mentioned seem impossible, perhaps we can fall back on the recent innovation, the MOOC! From the outside looking in, one might be tempted to think that MOOCs are the solution to equalizing participation in computer science education. Especially with class enrollments over 100,000 students. But on closer inspection, we soon learn that enrollments don't equal completions. Further, instead of equalizing participation, what MOOCs have done is amplify the underlying existing social structures (Toyama, 2011b). The majority of computer science MOOC takers already have a CS degree; which takes us back to where we started.

Heretofore, what MOOCs have successfully demonstrated is supplying continuing education in the CS space. While they are great at that, they have not successfully demonstrated their ability to bring in new participants into the CS pipeline. Since MOOCs mostly support

those who are already in the system, they do not necessarily increase the participation of ethnic minority students or female students; instead they may serve a retention role.

As a result, the general strategy is to focus further down in the computer science education pipeline to high school and drive computational thinking as the on boarding mechanism that will get people going and equalize participation.

Chapter 3

Research Methods

3.1 Formative and Mixed-Method Research

Research was conducted to understand the social and curricular cues that surround the decision of historically underrepresented students to advance in computer science at the undergraduate level at UC Berkeley.

As a woman of African descent, I fall into the broad demographic of the participants in my study; as a result, I have chosen an advocacy/participatory knowledge design approach (Creswell, 2003, pp.9) to investigate the socio-curricular issues that surround the decision to choose CS. Creswell states that in doing advocacy type of inquiry, the researcher co-creates with the participants the research instruments and may in the end join their voice with the participants in advancing the need for change. In his own words, he states thusly:

> "The "voice" for the participants becomes a united voice for reform and change. This advocacy may mean providing a voice for these participants, raising their consciousness, or advancing an agenda for change to improve the lives of the participants."

To test out the overall effectiveness of the *Beauty and Joy of Computing* (BJC) curriculum as implemented in UC Berkeley's CS10, in attracting historically underrepresented students as well as the specific effectiveness of the data lab on broadening students understanding of CS as scientific inquiry; a formative mixed-method research design was adopted.

Formative, because the outcomes of the research were used to improve the BJC curriculum, and mixed-method because both quantitative and qualitative approaches were implemented. To gain a comprehensive analysis into the socio-curricular effectiveness of the BJC curriculum as the first class in a student's CS trajectory, it was benchmarked against CS61A—the first class for majors, and increasingly, for non-majors as well. The quantitative approach gave the opportunity to have a landscape view on the effectiveness of the curriculum in attracting

the target audience, while the qualitative approach gave the opportunity to dig deeper into the specific lived experiences of the participants within the context of the classes under study. The research was constructed to answer the following research questions:

R1. What are the social factors that contribute to the underrepresentation of historically marginalized groups in CS?
R2. What impact does CS10 have in attracting female students into the major?
R3. What role does socio-culturally responsive curriculum play in attracting underrepresented students into computer science?
R4. How does a graphical-based programming language affect students sense of CS identity?

3.2 Data Collection

Observations, surveys and interviews were carried out to determine the effectiveness of the BJC curriculum in attracting historically underrepresented students into computer science. An overview of the research process is presented in table 3.1.

Formative Research: Blocks to Text Prototype Workshops

Formative research via participant observation was done at the Stanford middle school computer science program as well as at the UC Berkeley beyond blocks sessions. These observations were not part of the formal analysis, but they informed the design of the data lab and guided the creation of the CS10 Besides Block modules.

Based on the anecdotal feedback I had received from students who felt that drag-and-drop programming environment were not "real programming" environment, it became apparent that student's perception of graphical programming language may create an unintended obstacle further making the decision for students to transition into the major that much more daunting. As a means of investigating and addressing this issue, preliminary field work was done to prototype the concept of a transitional module from Snap*!* to Python, with the aim of helping students make the cognitive transition from realizing their code in a drag-and-drop environment to a text based environment.

Blocks to Text Prototype 1: EPGY

In the summer 2013, I created a transitional module from Snap*!* to C programing language, and tested it out in the context of a middle school computer science program for exceptional gifted and talented youth (EPGY) at Stanford University[1] Throughout the program,

[1]The EPGY is a residential summer program offered at Stanford University for high achieving students. The students came from Europe, Latin America and North America. The class size was 13 students, of which 7 were girls and 6 were boys. Their ages ranged from (11 - 13 years old). Half of the class had never programmed before they attended the summer class

Overview of Research Process

Date	Research Method	Content Focus
July 2013	Formative	Blocks to Text Prototype 1 – EPGY
November 2013	Formative	Blocks to Text Prototype 2 – UC Berkeley Beyond Blocks
February 2014	Quantitative	Design of Survey
February 2014	Qualitative	Design of Interview Protocol
May 2014	Formative	Design of Data Module
November 2014	Formative	Implementation of the Data Module within Lab Framework
November 2014	Quantitative	Administer Survey
November 2014	Formative	Administer Data Module
February 2015	Quantitative	Quantitative Analysis of Survey Data
April 2015	Quantitative	Administer Survey
April 2015	Formative	Administer Data Module
April 2015	Qualitative	Conduct Interviews
June 2015	Quantitative	Quantitative Analysis of Survey Data
July 2015		CSTA Meeting – Preliminary Findings Presented
July 2015	Qualitative	Qualitative Analysis of Interview Data

Table 3.1: Timeline of Research Process

I observed how students made the transition, and witnessed how Snap*!* was a great environment to grasp the big computational ideas, and made crafting solution in C (a text-based language) easier. This workshop ran for 9 days, of which the classes where 2 hours in length and the rest of the time devoted to students working on their own.

Blocks to Text Prototype 2: UC Berkeley's Beyond Blocks

Based on the observations that I made at the EPGY, I came back on campus and started investigating how UC Berkeley CS10 students were handling the transition. Fall 2013, I got together with some CS10 lab assistants and piloted a "Beyond Blocks" module for UC Berkeley students. It consisted of 5, 2 hour sessions that ran over 6 weeks. The modules were broken up into:

- session 1 - Mapping from Snap*!* to Python
- session 2 - In-depth Python

- session 3 - Data modeling module with Python
- session 4 - Web development (HTML, CSS, Javascript)
- session 5 - Web development (Flask Framework)

The workshop was held in an auditorium in the computer science building. The demand of this pilot and feedback from students is what prompted the decision to go to the BJC curriculum team and petition that Beyond Blocks become part of the official BJC curriculum. This desire was granted, and Beyond Blocks became part of the official curriculum fall 2014, where it was renamed "Besides Blocks" to highlight the fact the text-based programming languages are not superior in their computational power to blocks-based programming languages.

Quantitative Research: Surveys Instruments

Survey instruments were developed to measure participants' interest along several dimensions as is shown in table 3.2. This was done by extending previously validated instruments.

To measure students CS attitudes, previously constructed instruments from Titterton and Haynie were used (Titterton and Haynie, 2012).

To determine the role of identity and self efficacy; as well as the role of mentors in attracting underrepresented students, previously constructed instruments from (Martin and Scott, 2013) in their attitudinal study of CS in the Level Playing Field's Summer Math and Science Honors Academy (SMASH) were used.

Additional instruments were developed by the researcher to measure cultural competency. The survey uses a 5-point Likert scale (where 1 = Not Really, 3 = Neutral and 5 = Absolutely). A copy of the survey instrument is included in appendix E of this document.

Dimensions Developed to Measure Participant's CS Interest

Code	Dimension
atcs	Attitudes about CS competency.
atcsgender	Attitudes about the role of gender in computer science
atct	Understanding of computational thinking
blg	Sense of belonging in the CS classroom.
clet	Attitudes about social implications and ethics.
cltrcmp	Understanding around cultural competency.
mtr	Access to CS Mentors.
prcs	Pre-Collegiate computer science awareness.

Table 3.2: Survey Instrument Dimensions to Measure CS Interest

Analysis of Survey Instruments

To analyze the data collected in the study, I started off by filtering out missing data. All the data collected from CS10 over the two semester were merged together. For this study, only data items for which the students had entered either male or female was considered. Because the sample size of the data where students identified their gender as "other" was too small, it wasn't included in the analysis. 170 male students, and 217 female students consented to participate in the research from CS10, while 324 male students, and 171 female students consented to participate in the research from CS61A. An elaborated analysis process for the quantitative data collected in this study can be found in appendix B.

In order to test for statistically significance, a non-parametric alternative to the standard t test, the Mann-Whitney U test was used (Mann and Whitney, 1947). This decision was made because the study generated ordinal data through the use of a Likert scale, and no statements can be made with regards to the distribution of the data. Further, the Mann-Whitney test has been validated as acceptable when dealing with datasets for which there are unequal sample sizes. The significance threshold was set at 0.05.

The data was divided into sets as follows: Students who had previous pre-collegiate exposure to CS were separated from those that did not. This brought the sample size down from 882, to 480 for students without prior exposure to CS. The sample as distributed along gender and class as is shown in table 3.3.

Participants Breakdown For Analysis

Class	Female	Male	Female	Male
	With Prior CS		**Without** Prior CS	
CS10	45	48	85	101
CS61A	86	223	172	122
Total	131	271	257	223

Table 3.3: Participants Broken Down by Prior Exposure to CS

The data was normalized to a range from 1 to 100. This decision was made to allow ease of interpretation in terms of percentages.

Qualitative Research: Interviews

Along with the surveys, interviews were conducted to get a deeper sense of the effectiveness of the BJC curriculum in attracting historically underrepresented students. These audio-

recorded interviews were conducted at the university with participants that either attended CS10, CS61A, or both. Furthermore, participants were carefully chosen to reflect a broad spectrum of computing experiences from novice first time beginners to more advanced CS students.

I conducted individual, 30-45 minute, semi-structured qualitative interviews with participants. A copy of the interview protocol that was used is included in appendix D of this document.

During these interviews, I asked participants about their personal history with their use of computers, about their academic interest(s), their perception(s) of CS, their reason(s) for taking the class, and their experience around it.

These interviews were taped and later transcribed. Analysis was guided by a grounded theory approach. Interview text was read first to identify emergent themes.

3.3 Location and Context

There are three ways a student can take to become introduced to CS at UC Berkeley. The student may choose to take CS10, or chose to take CS61A, or instead choose to take CS61AS—The self-paced version of Structure and Interpretation of Computer Programs. For the purposes of understanding the experience of undergraduates students around introductory CS, the research context focused primarily on CS10 and CS61A.

Class Context

The two classes under evaluation to determine their effectiveness in broadening participation in computing are held every semester at the UC, Berkeley. CS10 has a class size of approximately 200+ students each semester, while CS61A has an approximate class size of 1000+ students. Participation in both classes are continously growing at the university. As of the writing of this study, CS10 has a near 50-50 gender breakdown between male and female students, while CS61A's gender breakdown is approximately 34% female and 66% male.

3.4 Participants

The participants that were part of this evaluation came from CS10 and CS61A. These were mostly undergraduates whose demographic skew strongly towards White and Asian students. Surveys were conducted with 882 participants, while interviews were done with 24 participants. Participants where recruited for the interviews based on their willingness to participate in the study from their responses to the surveys. Tables 3.4, and 3.5 show the breakdown of the participants in the study.

Survey Participant Break Down

Class	Female	Male
CS10 Fall 2014	144	116
CS10 Spring 2015	73	54
CS61A	171	324
Total	388	494

Table 3.4: Survey Participants

Participant Recruitment

To recruit students who were willing to take both the survey and participate in interviews, a two-tiered recruitment approach was taken. First the survey was digitized and made available online via UC Berkeley's Google Apps for Education, Google Forms.

For CS61A, survey participants were recruited as follows: The course instructor sent out an online survey link to the students in the class, asking them via email for their participation in the research.

For CS10, survey participants were recruited as follows: The online survey was embedded in the first python lab module that all students are mandated to take. Participation in the survey was made optional for students. However, to incentivize them to take it and add their names as a signifier of willingness to be interviewed, some extra credit points were awarded to students who did. The text outlining students voluntary participation is shown below.

> This survey is to get some feedback about how everyone feels about programming. We have a new unit this semester, and your responses will be very helpful! It's going to be worth a small amount of extra credit if you're willing to put your name at the end.

For both CS10 and CS61A, participants for the interviews were recruited via email that I sent directly to students who had indicated their willingness to further participate in the study via entering their names in the surveys. The body of the email solicitation is shown in table 3.6. Students that responded were asked to book convenient times for them to be interviewed via an online scheduling service set up at https://omoju.youcanbook.me/. Interview participants were given stickers and chocolate as a show of gratitude.

Interview Participant Break Down

Class	Gender	Race	Pseudonym
CS10	Female	Asian	Django
	Female	Black	Rapunzel
	Female	White	Willow
	Male	White	BeanStalk
CS61A	Female	Black	Padme
	Female	Asian	Serena
	Female	White	JaneEyre
	Female	White	Uhura
	Male	Black	Imperator
	Male	Black	JarJar
	Male	Black	Titan
	Male	Black	Superman
	Male	Asian	BobaFett
	Male	Asian	ObiWan
	Male	Asian	Chewbacca
	Male	White	Chicora
	Male	White	HanFoldo
BOTH	Female	Asian	Saturn
	Female	Asian	Earth
	Female	Asian	Pluto
	Female	White	Uranus
	Female	White	Venus
	Female	White	Gigi
	Male	White	Neptune

Table 3.5: Interview Participants' Demographics

Hi there,

My name is Omoju Miller, and I am a doctoral research student in Computer Science Education here at UC Berkeley. I have been part of the CS10 BJC curriculum family for a few years now.

I love Computer Science a lot and was rather surprised that I was one of the few female students at my undergrad, then at work as a software engineer. So I decided to do something about it! I came to Cal to get my phd in CSEd, so I could study the factors that are causing people like me to not pick CS. You can help me a lot by agreeing to be share some of your experiences around CS so far.

I would like to interview you for no more than 30 minutes to learn about your experience in [appropriate class], as well as your experience in CS at Cal so far.

My schedule is very flexible, I am on campus at Sutardja Dai Hall most days of the week. I would need no more than 30 minutes of your time.

The data you give me will help make the department better. I want to assure you that this will be anonymous, and I will not publish personal identifying information about you.

Help me make EECS even more awesome!
Cheers

Table 3.6: Participant Solicitation Email

Chapter 4

Design of an Inclusive CS0 Course

4.1 The Beauty and Joy of Computing - CS10

As previously noted, once the College Board pulled the CS Advanced Placement (AP) AB examination from its roster in 2008, there was a scramble to use the vacuum that was created as an opportunity to re-invent introductory computer science for inclusion. The parties involved faced two inter-connected challenges. The creation of a new course that would potentially maximize participation and equally important the addition of 10,000 CS teaching staff at the high school level. To address the former, NSF in partnership with College Board created a committee to invent a new CS Principles course. While to address the latter, NSF created its CS10K project tasked with the engagement and preparation of 10,000 new CS teachers, in 10,000 high schools teaching the new curriculum by 2015 (Astrachan et al., 2011b).

To ground our conversation, lets take a quick look at the major challenges facing the maximization of computing. As a response to the challenges, two teaching professors at UC

1	A lack of presence of CS in K-12 education.
2	The under-production of post-secondary degrees in CS.
3	The underrepresentation of women in CS.
4	The underrepresentation of ethnic minorities in CS.
5	A lack of positive CS role models in the media.

Table 4.1: Major Obstacles to equalizing Participation in Computer Science Education

Berkeley, Brian Harvey and Dan Garcia seized upon the opportunity to create a new CS principles CS0—a CS curriculum designed specifically for non-majors—curriculum titled the "Beauty and Joy of Computing," (BJC) (Garcia et al., 2012; Garcia et al., 2014). The curriculum is offered at UC Berkeley as the class "CS10." A version of BJC is also taught at

the University of North Carolina Charlotte, that infuses some elements of video game design into the curriculum. In their own words, the creators of BJC declare:

> "BJC invokes passion, beauty, joy, and awe through engaging students in a rigorous computing curriculum that promotes creativity and collaboration using *Snap*'s visually rich programming environment, while also provoking thought around current events and how computing relates to people's lives." (Garcia et al., 2011)

The BJC curriculum was inspired by middle and high school Scratch-based CS summer programs that was designed by Colleen Lewis while she was a graduate student at Berkeley (Lewis, 2014). The Lewis curriculum had shown promise in attracting underrepresented ethnic minority students. Every aspect of the curriculum has been designed to be inviting. Even the logo and the mascot of the curriculum are friendly and inviting as can be seen from figures 4.1 and 4.2 respectively.

What makes the BJC curriculum fundamentally different from other introductory to CS curriculum is its emphasis on the "Big Ideas" of computing like: *functions-as-data*, *functional programming*, *recursion*, *concurrency*, *simulations*, and the *limits of computation*; ideas that are fundamental to core of computer science.

Furthermore, the BJC curriculum presents these ideas in the context of society through its social implications of computing thread. A strong emphasis is placed on how computing impacts society. This thread is woven throughout the entire course, helping students understand the unintended consequences of computing that can leave a lasting impact on the social, ethical, and economic fibers of society.

While BJC cannot on its own address all the major obstacles to equalizing participation in CS as listed in table 4.1, it does focus on increasing presence of CS in K-12 through its creation of a professional development summer program with the aim of teaching the BJC curriculum to in-service high school teachers. Further, the curriculum was written explicitly to broaden participation of historically underrepresented groups in computing.

Intro CS Track, UC Berkeley

At UC Berkeley, there are two separate ways a students can get a CS degree. They can either get a *Bachelor of Arts (B.A.)* through the College of Letters and Sciences (L&S), or get a *Bachelor of Science (B.Sc.)* through the College of Engineering. The major difference between the two tracks is that students who get the B.A. get to take breadth requirements that gives them exposure to more of a liberal arts education.

Once a student has decided that they might be interested in trying out CS, there are three major pathways the student can take. They can either take CS10, or CS 61A, or CS61AS—

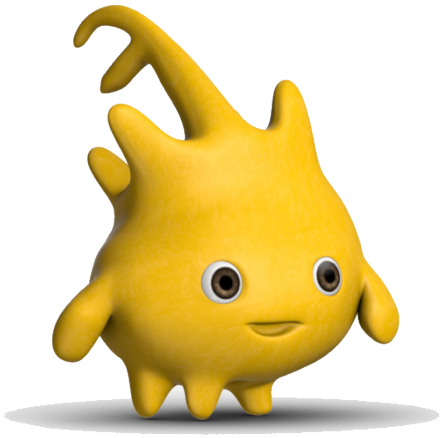


Figure 4.1: Alonzo, the mascot of CS10.

Figure 4.2: The BJC Logo.

the self-paced version of CS 61A. Table 4.2 shows the major differences between the three classes.

Having taking one of these classes, if a student then decides they are interested in either majoring/minoring in computing, whatever the track that student is in, they have to take the computer science "61 series." This series comprises of three classes:

- CS61A - Structure and Interpretation of Computer Programs.
  TIntroduction to programming and computer science. This course exposes students to techniques of abstraction at several levels: (a) within a programming language, using higher-order functions, manifest types, data-directed programming, and message-passing; (b) between programming languages, using functional and rule-based languages as examples. It also relates these techniques to the practical problems of implementation of languages and algorithms on a von Neumann machine. There are several significant programming projects.[2].

- CS61B - Data Structures.
  Fundamental dynamic data structures, including linear lists, queues, trees, and other linked structures; arrays strings, and hash tables. Storage management. Elementary principles of software engineering. Abstract data types. Algorithms for sorting and searching. Introduction to the Java programming language.

[1]The data is valid as of fall 2014
[2]Taken from the UC Berkeley course catalog

	CS10	CS61A	CS61AS
Format.	Lectures (with some guest lectures by research faculty), discussions and labs.	Discussions and labs.	Lab-centric and discussions.
Time Commitment.	Four units.	Four units.	Variable, a student can take one to four units over one or two semesters.
Counts Toward Major?	No.	Yes.	Yes.
Need for Prior-programming Exposure.	CS10 is designed for students without previous programming experience.	There is no formal programming-related prerequisites for admission to 61A. However, students without prior experience typically spend a large amount of time each week on the course.	CS61AS has an optional "unit 0" designed for a gentler introduction to CS and programming.
Programming Language(s).	– Snap*!* – Some Python	– Python – Some Scheme – Some SQL	– Scheme/Racket – Some Python – Some Logic
Rough Class Size[1]	300	1200+	100

Table 4.2: UC Berkeley's Intro CS Class Paths Deconstructed

- CS61C - Machine Structures.
  The internal organization and operation of digital computers. Machine architecture, support for high-level languages (logic, arithmetic, instruction sequencing) and operating systems (I/O, interrupts, memory management, process switching). Elements of computer logic design. Tradeoffs involved in fundamental architectural design decisions.

Before the BJC curriculum was invented, the introductory course that was designed to serve non-majors was titled: "Computer Science 3: Introduction to Symbolic Programming (CS3)". The course was created to help students get ready to succeed in CS61A, the first CS class designed for majors. As a result, it used the same programming language that was used in CS61A, Scheme—a funtional style, text-based language.

In her study of attrition in undergraduate CS at Berkeley, Lewis found that female students were disproportionately weeded out of the track, often starting at CS3 (Lewis, 2010a). For those students who had no prior programming background—majority female—CS3 served as a gateway to get them ready to succeed in the 61 series. The Lewis 2010 study used the CS3 online curriculum database to analyze the attrition patterns for 14 semesters, from the fall of 2002 to the spring of 2009. According to Lewis:

> "When controlling for level in school, major and the semester and year the course was taken, the odds of a female student dropping the course is 32.0% higher than for a male student."(Lewis, 2010a)

Over her 14 year span of the data, she found the drop-out rates for female students was 27% while that of male students was 20%. Further, she noted that certain semesters like Fall 2006, female students dropped out of the class by 46% while only 18% of their male counterparts dropped the class. Furthermore, she found when controlling for the level in school, the major, and the semester/year the course was taken; the odds of a female student dropping the course was 32.0% higher than for a male student.

In the past, most introductory to CS curriculum took a narrow view and placed priority on teaching the skill of programming. There was not a lot of emphasis placed on motivation, or the impact of technology on society. CS3 curriculum adhered to that paradigm, it was devoid of computing in the context of society. A student taking the CS3 curriculum would not have the opportunity to connect their CS knowledge to the human beings that created the knowledge, to the conditions that enabled the creation of the knowledge, and to the implications of the knowledge. It was a sterile approach that chugged computation down the throats of the students.

The BJC approach fundamentally deviates from that by placing computational knowledge as a tool that should be in service of society. In addition to learning about conditionals, recur-

[3]The essay was later changed to a one page blog submission

	CS3	CS10
Emphasis	– Funtional Programming	– Functional and Imperative Programming – Video Games – 3D Graphics – Concurrency – Distributed Computing – Social implications of computing – Applications that changed the world – Computing in Industry – Saving the World with Computing (CS + X) – Higher Order Functions and Lambda – Cloud Computing – Game Theory – Artificial Intelligence – HCI –
Programming language	Scheme	Snap*!* with a week of Python at the end.
Projects	One big project	Two projects and an essay[3].

Table 4.3: Major Deviations between CS3 and CS10.

sion, higher-order functions and so on, students are also introduced to contemporary issues at the intersection of CS and society. Table 4.3, list the major ways in which the CS10 curriculum deviated from the CS3 approach. Additionally, in order to maximize participation, the BJC curriculum was designed with the following constraints:

Design Constraints of CS10

- Even though CS61A does not have an explicit pre-requisite for prior programming exposure, it is designed at a pace that works best for students who have had some prior exposure. Furthermore, CS61A delves deep into the nature of recursion—a computational problem-solving method that is difficult for the novice to grasp—as a result CS10 should highlight it.
- Since the approach is fundamentally different from existing introduction to CS curriculum, students in BJC should know:
  - CS History.

  - They should be exposed to the CS + X approach, i.e., CS + X for all X[4].
  - Applications that changed the world.
  - Summaries of interesting research areas, like Artificial Intelligence and Human-Computer Interaction.
  - Computing de-mystified and an understanding that computing is really fun.
  - Half of the course should be the societal implications of technology. Table 4.5, lists some of the issues spring 2015 instance of CS10 dealt with.

- Ensure an environment that engenders passion, beauty, joy and awe of computing.
  - Taking every step to ensure that the course stays attractive to historically under-represented groups.
  - Prioritize the need for students to decide their own projects.

- Make sure the entire course is free to students, so that the class can be exported to high schools easily.

- Since deep learning happens by doing, and not by listening. As a result, the BJC curriculum uses a lab-centric model of instruction.

CS10 has been offered at UC Berkeley for Since 2010. In the 2010 pilot alone, 43 of the 77 college students chose to continue to the next more demanding first course intended for CS majors, CS61A.

In 2010, UC Berkeley became one of the five pilot sites to implement a version of the CS Principles course that had been developed in line with the curriculum framework of College Board's AP CS principles' seven big idea of computing and its accompanying six computational thinking practices. The framework is laid out in table 2.3 (Astrachan et al., 2011a). Table 4.4. lists the other four schools that were part of the 2010 pilot.

From table 4.4, one can easily see that UC Berkeley's version of BJC far exceeds the contact hours of the remaining four classes in the pilot. The reason for this is that the class is made up of four hours of lab, two hours of lecture and one hour of discussion weekly for 14 weeks. As of the spring semester of 2015, these are the topics that are covered in the class:

Topic 1: Introduction to Snap*!*

[4]Where X stands for an unknown subject like Biology

[5]University of North Carolina, Charlotte. UNCC, which taught the course in the Spring of 2011 following all four others' offerings in the Fall of 2010, decided to model their course after BJC and joined as a partner.

[6]UC Berkeley

[7]Metropolitan State College of Denver

[8]University of California, San Diego

[9]University of Washington

School	Name	Programming Language(s)	Size	Contact Hours
UNCC[5]	The Beauty and Joy of Computing	– Scratch – GameMaker	30	45
UCB[6]	The Beauty and Joy of Computing	– BYOB	76	98
MSCD[7]	Living in a Computing World	– Scratch	20	60
UCSD[8]	Fluency w/Information Technology	– Alice – Excel	575	50
UW[9]	CS Principles	– Processing – Python	40	50

Table 4.4: Comparison of the 2010/2011 Pilot courses, (Astrachan et al., 2011a).

Topic 2: Loops and Building your own Blocks
Topic 3: Conditionals and Variables
Topic 4: Advanced Building: Abstraction and Testing
Topic 5: Lists
Topic 6: Algorithms
Topic 7: Algorithm Complexity
Topic 8: Finch Robots
Topic 9: Concurrency
Topic 10: Trees and Fractals using Recursion
Topic 11: Recursive Reporters
Topic 12: Tic Tac Toe
Topic 13: The Internet
Topic 14: Practice with HOFs and Functions as Data
Topic 15: Besides Blocks: Welcome to Python
Topic 16: Besides Blocks: Data Structures in Python
Topic 17: Besides Blocks: Data in Python

BJC Hits the Big Time!

The fall 2010 offering of BJC at Berkeley was extremely successful with female students who comprised 45% of the class—with this enrollment rate equal to the 11-year high in the previous (programming-only) version of the non-majors course, CS3 (Lewis, 2010a).

By the spring semester of 2014, the class reached a significant milestone that sent ripples

reverberating throughout CS education, and in the media. That spring, I happen to be part of the Berkeley contingent that was attending the Richard Tapia Celebration of Diversity in Computing conference in Seattle. At Tapia 2014 conference, Dan Garcia was one of the plenary speakers. He presented on the ongoing efforts around the BJC curriculum. It was during that presentation that he mentioned the milestone statistic of women outnumbering men in the intro CS class designed for non-majors. For as long as statistics had been kept about class enrollments—which go back to 1993—that semester, CS10 had more female students enrolled than than male. What happened next was like magic.

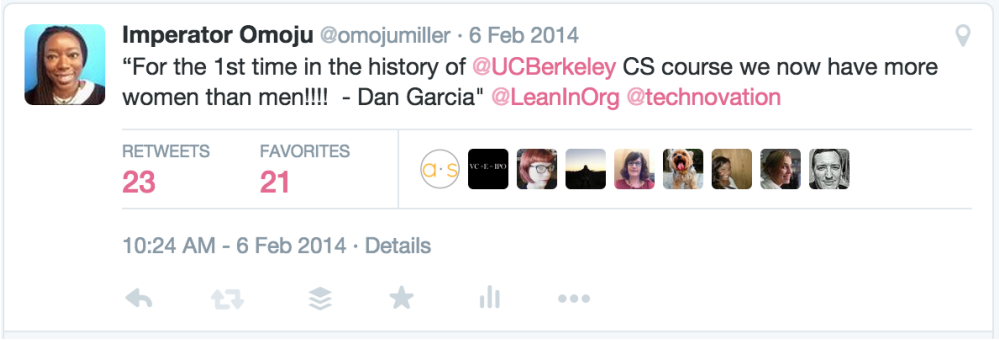


Figure 4.3: The Tweet that Landed CS10 on the Front Page of the San Francisco Chronicle.

During Dan Garcia's presentation, I sent out the tweet displayed in figure 4.3, I made sure to copy LeanIn—Sheryl Sandberg's high visibility foundation, and the Technovation challenge—a women and girls app design contest. Between LeanIn and Technovation, the tweet and its information landed CS10 on the front page of the San Francisco Chronicle (Brown, 2014). From there on versions of the story were reprinted on several media outlets. The story even landed in TechCrunch, silicon valley's technology blog of choice (Ferenstein, 2014). By December of that same year, the White House in a press release announcing its "New Commitments to Support Computer Science Education," singled out BJC's recent collaboration with New York City schools as an example of the new commitments that were being made to support CS education in the K-12 space (The White House, Office of the Press Secretary, 2014).

4.2 BYOB to Snap*!*

> *"... a rigorous graphical language that both kids and computer scientist can use Snap!"*
>
> — (Harvey and Mönig, 2010)

Apart from its re-energized curriculum, one can argue that another reason why CS10 has been successful in attracting students is its use of Snap*!* graphical, blocks programming

language. When the Scratch programming language was released, many educators adopted it as their go-to language for teaching introductory computer science because it was very friendly, one of the major reasons why it worked so well for children. After all, Scratch was descended from LOGO. But the first version of Scratch was designed in such a way that intentionally prevented the creation of procedures. The designers decided they didn't want children to get stuck trying to grapple with abstract concepts. That design decision made it difficult to use Scratch as a language for a rigorous computer science class.

Luckily, the software engineer Jens Mönig took it upon himself and built an abstraction layer on top of Scratch that allowed the user to build their own blocks. This extension of Scratch was christened BYOB (Build Your Own Blocks). Once this extension was created, BYOB became an appropriate programming language to use for Berkeley's CS10.

With the creation of BYOB, the first pilot of CS10 was launched in 2009. As part of the CS10k effort, the BJC team works with high school CS teachers to test out the curriculum in the high school space. These teachers encountered some negative feedback because the programming language's name could be mistaken as the acronym for Bring Your Own Booze. As a result, BYOB was renamed Snap*!* (Harvey and Mönig, 2010; Harvey et al., 2014).

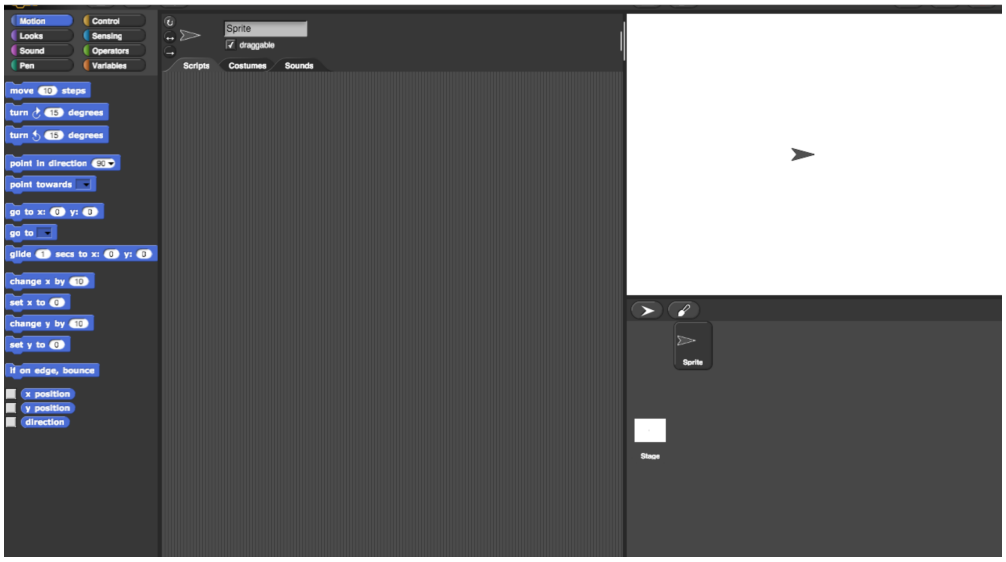
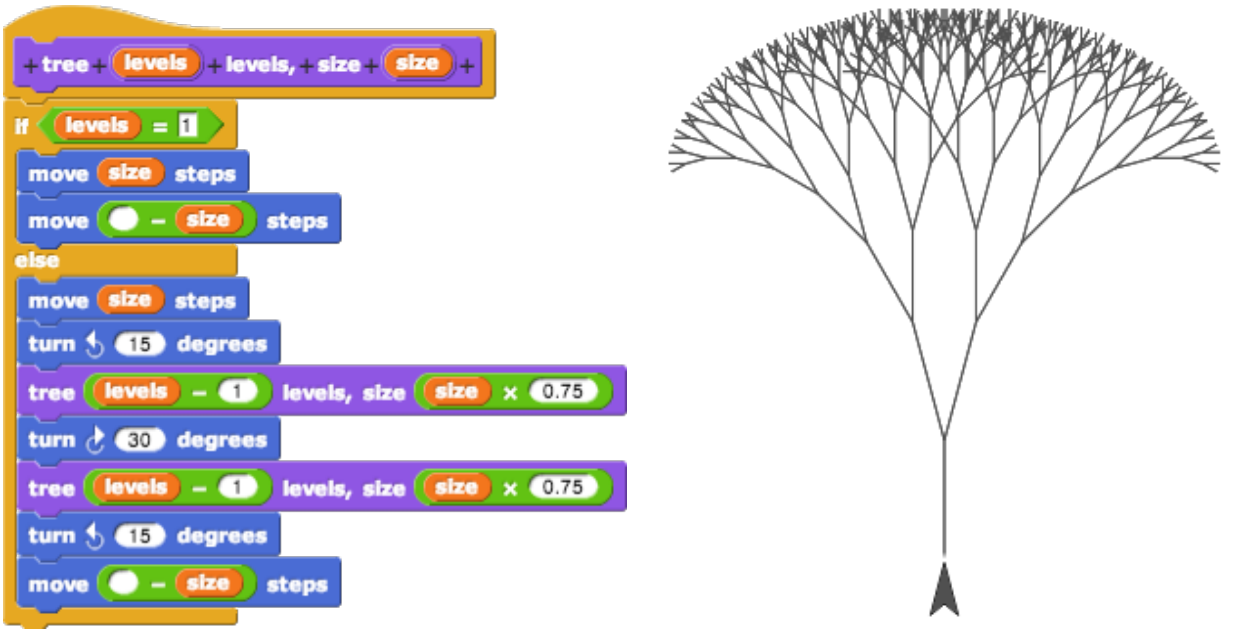


Figure 4.4: The Snap*!* programming environment.

Snap*!* is built to work in most web browsers, it is free and hosted by UC Berkeley. Figure 4.4 shows the in browser environment of Snap*!*, while figure 4.5 displays a Snap*!* script that does recursion, and the resulting tree that is drawn on the stage as a result of the recursive call.

The Snap*!* environment is divided into 3 columns. The first column hold a set of color coded primitive blocks that are categorized by their main functions. The middle column is



(a) Snap*!* Script. (b) The output of Snap*!* Script.

Figure 4.5: Snap*!* Script, and Result of Snap*!* Script.

the area that selected blocks for computation are put. The final column holds a stage for visual display and a sprite menu.

The Challenge of Graphical Languages

While Snap*!* had in fact achieved being a rigorous programming language capable of demonstrating higher order functions, students in general did not believe that programming in Snap*!* was "real" programming. Even though students could do "map-reduce," a computational algorithm that was invented by Google to help scale computation over large amounts of data—programming does not get realer than that—students were still not convinced what they were doing was the same thing that programmers did in industry, but just realized in a different medium. The reason why this is important is that if students do not believe they are programming, then they are no closer to identifying themselves with the field.

Lets take a step back. What is arguably the first *educational* programming language, LOGO, was designed based on command prompts and text, as you can see from figure 4.7. Even though it was designed specifically to serve a pedagogical purpose, it was realized in the same visual substrate as the *professional* languages of its time. Furthermore, no parts of LOGO was *dumbed down* for the sake of children. LOGO came built in to support procedures, children learned to write their own procedures and even did recursion. Research showed that students who were introduced to LOGO did not suffer from the foul misunderstanding

that they were not engaged in real computer programming (Lewis, 2010b). Lewis found that "students that learned Logo had on average higher confidence in their ability to program," than their counterparts that learnt Scratch.

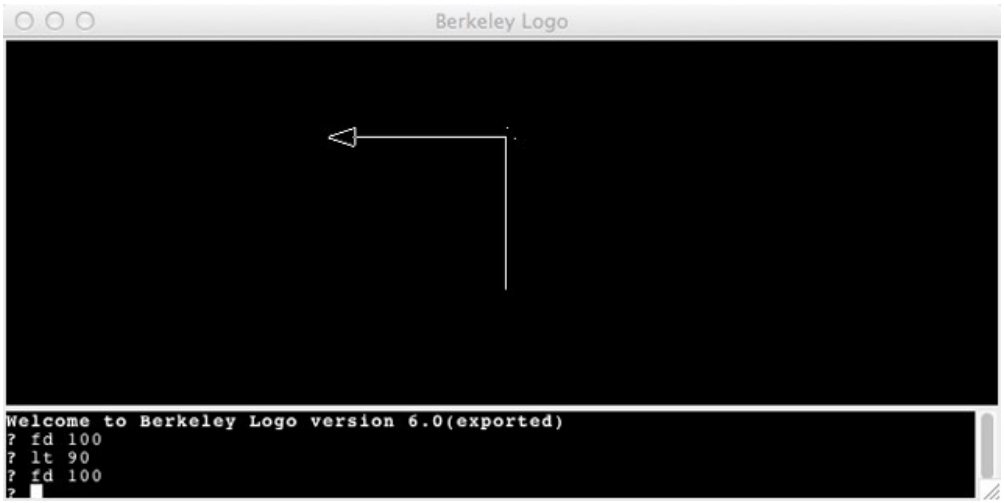


Figure 4.6: The LOGO programming environment.

Continuing in the same line of inquiry, Lewis posed the question (Lewis et al., 2014):

1. Does completing a worksheet that shows how the programming language Scratch relates to Java, C++, and Python:
   - Increase students' confidence and interest in computer programming?
   - Convince students that Scratch is a programming language and will help them when learning other programming languages?
2. What do students think counts as a programming language?

For the first question, they found the answer to be inconclusive. The intervention was designed specifically to help students understand that Scratch was a valid programming language. Even though students often answered yes when asked the question if Scratch was a programming language; however, when students were asked if other blocks-based graphical programming environments like AppInventor and LegoMindstorms were programming environments, they often said no.

While this idea is problematic, it helps to reveal something about what students attune to with regards to learning computer science. Similar findings has been documented in existing literature that shows that African American male students also felt that programming in a visual drag-and-drop programming language like *Alice* did not feel like "real" programming as opposed to using a text-based Python variant(DiSalvo, 2012, pp. 113-120).

We have found similar results in CS10, data from post course surveys showed that students did not feel that programming in Snap*!* was "real" programming. Interviews with students who have advanced on to CS61A revealed that they felt that there was a need for a bridge unit between programming in a drag-and-drop environment to Python, the text-based environment that is used in the next class.

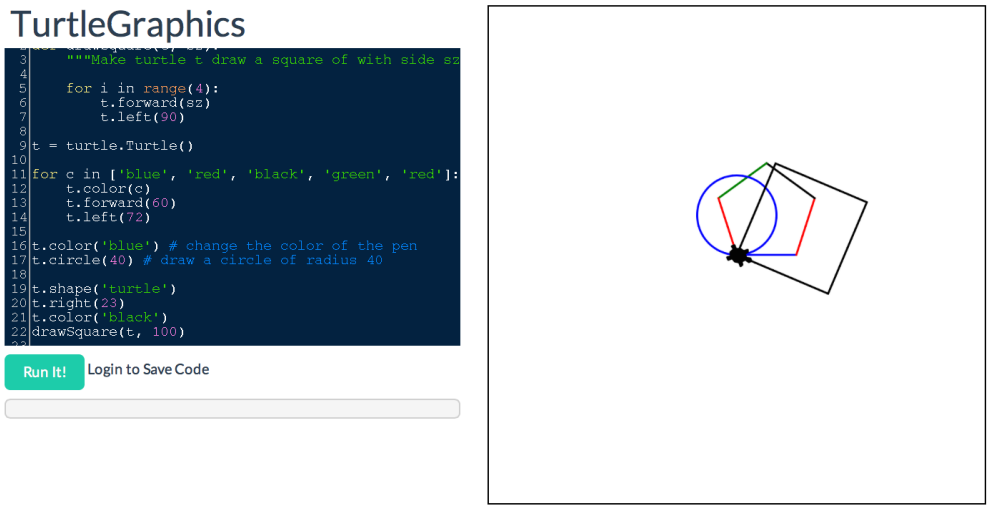


Figure 4.7: Text Based programming in the browser

Subsequently, students who have advanced to the next class in the series may have a hard time transitioning into the class as we shall see in the findings of this research. Furthermore, this challenge of not identifying blocks languages as professional is exacerbated by the fact that female students may take CS61A at a time when they are beginning to contemplate exiting the major all together.

As a result, the belief in the creation of a responsive learning unit, perhaps based on the "codification" project that shows how to map Snap*!* code to Python, Java, Smalltalk and C, Which bridges both the visual language and text-based language is worthy of investigation.

4.3 Social Implications of Computing

> *"Until the lions have their own historians, the history of the hunt will always glorify the hunter."*
>
> — Anonymous.

As we can see from table 4.1, one of the challenges to equalizing participation in CS is the underrepresentation of ethnic minorities. We often see this fact mentioned, but often not

the reason why we should care about it. In my opinion one of the reasons we should care about this fact is the role that data, statistics and now machine learning play in the lived experiences of Black and Latino youth in the United States.

While machine learning and data mining have played an important role in enabling innovation, from the creation of IBM's Watson computer that is aiding medical diagnosis to Google's stellar recommendation engine. Underlying these innovations is the massive aggregations of user-generated-data, and *that* is a double-edged sword—as we are starting to see with the rise of "so-called" racist algorithms. If you google "racist algorithms," you get over 200,000+ hits, of which the first is a Washington Post story covering Google Maps' unfortunate experience around the White House, where some users had labeled the location as "Ni***r House" (Dewey, 2015).

Interactions like these, i.e., computing in the context of society, is what the social implications thread woven through CS10 is designed to help students understand. Table 4.5, shows some of the readings that were discussed as part of that thread in the Spring 2015 session of CS10. These readings help elaborate on ideas around privacy, ethics, discussion of the role of patents and intellectual property, censorship in the context of pornography, professional standards of ethics and so on. The lack of understanding of these very topics by software engineers—a failing of the engineering centered curriculum that has whittled away the liberal arts—is beginning to compound the effects of institutional discrimination as we shall see in the following discussion.

Software	Marc Andreesen: Why Software Is Eating the World.
Digital Games	Louis Von Ahn and Laura Dabbish: Designing Games With a Purpose.
	Adam Liptak: Justices Reject Ban on Violent Video Games for Children.
Digital Citizenship	Douglas Rushkoff: Program or Be Programmed: Ten Commands for a Digital Age.
	Samuel Greengard: Living in a Digital World.
Robots	Jaron Lanier: The First Church of Robotics.
Security	Duncan Davidson: When Servers Bleed: A Tale From the Front Lines of Dealing with the Impact of the Heartbleed OpenSSL Vulnerability.
Data	James Temple: Web 2.0 Summit: Data Explosion Creates Revolution.

Table 4.5: Social Implications of Computing Topics

To motivate the importance of examining the impact of computing on society, especially the unintended consequences of computing, I will share a contemporary example that has to do with machine learning, i.e., crime prediction or what I like to think of as the history of race-based policing and the invention of the idea of "Black Criminality."

The Biography of an Idea: Black Criminality

In a 2012 episode of the television news journal show *Moyers & Company*, in conversation with the historian Khalil Muhammad, Moyers remarks:

> "...our brethren at WNYC, the public radio station here in New York, recently ran a series in which they reported that one in five people stopped last year by New York City police were teenagers 14 to 18 years old. 86% of those teenagers stopped were either Black or Latino, most of them boys. Last year, more than 120,000 stops of Black and Latino; the total number of Black and Latino boys that age in New York City isn't much more than that. 177,000 or so, which suggests that every teenage boy who's Black and Latino in this City of New York is likely before he graduates to have been stopped and frisked by the police." (Moyers and Muhammad, 2012)

If one rejects the myth of biological determinism, then one has to conclude that is highly improbable that all Black and Latino youth in New York City will be engaged in criminal activity. Muhammad responds to Moyer's by outlining the biography of the idea housed in the notion of Black criminality. He traces its inception back to the end of the civil war and the beginning of reconstruction when the national demographers had to start including the recently freed Black peoples in their projections for the allocation of resources.

The idea of Black criminality came to be, based on the unintended consequences of the Uniform Crime Report; the report that is the official data on crime in the United States, routinely published by the Federal Bureau of Investigation (FBI) (The Federal Bureau of Investigation, 2006). Before it was conceived in the 1930s, cities collected data around crime based on ethnicity. Cities like New York had their own reports that tracked crime along ethnicity, the data was collected on Italian, Irish, German, Scandinavian, Mexican, Blacks and so on. Once the effort was nationalized by the invention of the Uniform Crime Report, the dimensions on which crime was tracked was collapsed into Whites, Blacks, Foreign Born, and Other. A few years later, those same dimensions were further reduced to White, Black, and Other.

The seemingly innocuous acts of dimensionality reduction, probably taken to reduce complexity had the unintended consequence of understanding crime based on a White norm. It in effect made the statistical definition of crime in the United States be the deviation from a "White" norm. On the invention of Black Criminality, Muhammad states,

> "So as long as Blacks in that accounting showed disproportionate levels of any activity across those categories, White was always normalized. And in effect, it made invisible White criminality. We don't talk about White criminality. We don't talk about the White prison population." (Moyers and Muhammad, 2012)

Now, these same crime statistics that help invent the idea of "Black criminality" are been called upon to help predict crime in places like Los Angeles, San Francisco and other major metropolitans areas (Berg, 2014). Machine learning algorithms have been crunching on these kinds of data to help build software systems that are called "predictive policing" systems; creating an ever more compounding effect of the unintended consequence that some demographers made through a seemingly innocent act of dimensionality reduction.

The Importance of Social Implications of Computing

As I have outlined in the previous section on the invention of Black criminality, we can see that either human bias, or the unintended consequences of human behavior can be compounded and institutionalized by the use of algorithms that depend on those pieces of knowledge.

Just like with most things, they are not in of themselves moral or immoral, it is the human intention behind it that makes a piece of technology liberating or constricting. The social implications of computing seek to help students understand that their ways of knowing can have a profound effect on society; especially a society that is increasingly dependent of the artifacts of computing.

The good news is there are now respected members of the software engineering community who are speaking up about the importance of studying the culture of computing just as much as computing itself. Jeff Atwood in an interview states it best,

> "Programming at it core is a very human activity, and a lot of it is determined by how you treat the other people on the project and how they treat each other ... Computers change all the time. That is true, we know that to be true. But people do not. People do not change that much over time. Studying the people, and how they interact with other people will get you that much further in your career than being an assembly language genius." (Atwood, 2015)

Perhaps as the criteria that we use to quantify what makes a good computing citizen slowly shifts to incorporate good interpersonal skills, this will naturally lead to a more roburst understanding of the social implications of computing.

For marginalized persons in particular, highlighting the social implications of computing can be a very powerful tool in helping then connect with CS in a personal way, and harness its computational affect to tell their own story.

Chapter 5

Design of Hip-Hop Data Module

> *"We build computers and programs for many reasons. We build them to serve society and as tools for carrying out the economic tasks of society. But as basic scientists we build machines and programs as a way of discovering new phenomena and analyzing phenomena we already know about. Society often becomes confused about this, believing that computers and programs are to be constructed only for the economic use that can be made of them."*
>
> — A. Newell and H. Simon

One of the approaches that has been suggested to help maximize participation in CS, is an emphasis on interpreting and using information via computational artifacts. During its ideation process for the invention of the new AP CS principles framework, the College Board toyed with the idea of including a data project as part of a portfolio of work students created to demonstrate their knowledge of computing. Unfortunately, they later decided to exclude it.

Even though College Board eventually decided against the inclusion of a data module in students' portfolios, the learning objectives remained in place on the importance of understanding data manipulation techniques as a fundamental skill, like learning objective LO.3.2.1, which states, "extract information from data to discover and explain connections, patterns, or trends" (College Board, 2014). There is also a growing movement in academic cirles about the importance of data science.

While data manipulation techniques are important, they are a means and not an end; the end is gaining knowledge and insights out of data. My friend and esteem colleague, John Denero, puts it best in his introduction of a new undergraduate course "Foundations of Data Science" at UC Berkeley, Fall 2015. He states his rationale for creating the course as follows:

> "We wanted to put together a set of ideas that will allow people to make informed decisions using data in this data rich world in which we live" (DeNero, 2015).

The power of data manipulation techniques come to light when they are directed in service of CS as scientific inquiry. These techniques allow the computer scientist to transform into a researcher, asking questions and creating epistemology about society; finally giving us the opportunity to build a bridge between intuitive understanding and formal knowledge.

This data module was specifically designed to help students understand the role of computer science as inquiry by guiding them through a scaffolded series of activities which investigates a "Hip-Hop" way of sense-making through the computational exploration of rap lyrics. In addition, the philosophical assumptions of this research has been the notion that education/learning should have cultural resonance, should be "fun," and should be deeply embedded in society. Together these assumptions help guide the design of the Hip-Hop data module (Miller, 2014). The Hip-Hop data module as been implemented in the BJC curriculum as the final lab exercise of CS10.

5.1 Computer Science as Scientific Inquiry

In my opinion, one of the major reasons why students have a difficult time wrestling with data in the context of CS, is a lack of understanding of CS as scientific inquiry. To put it more precisely, a failure to grasp the *science* of computing.

As part of this research, I have spent time asking undergraduate CS students to tell me the difference between a "Computer Scientist" and a "Programmer." It has been interesting but not surprising to learn that most students don't know what distinguishes these two titles.

I believe the fault sits squarely on the shoulders of the academy. In many institutions, the CS department sits under the departmental structure of the Arts and Sciences, and in other institutions, its seats under the college of Engineering. The reason for this is that computer science is both an *art* as well as a *science* (Knuth, 1974; Roy and Haridi, 2001). Most students don't think of the field in this way, the idea of CS as a science is not relatively new; however, it isn't explicitly discoursed.

So if CS is both an art and a science, then what exactly is the *art* of computing, and what exactly is the *science* of computing? In his essay *Computer Programming as an Art*, Donald Knuth posits that *art* in this sense is the application of knowledge, whilst *science* as it is used in computer science stands for knowledge, i.e., what we know (Knuth, 1974). What Knuth doesn't mention is the third leg of this knowledge stool, the question of how we know? And that, is acquired through the process of scientific inquiry.

The curricular intervention that is at the heart of this research speaks to the notion of computer science as scientific inquiry (Newell and Simon, 1976). The great leaders of computer science—Herb Simon and Allen Newell—believed that CS was an experimental science, that

every machine and in our case, every program written is an experiment that poses a question to nature and its behavior offers clues to the answer. Our hypothesis in answering these questions are what guides us in our construction of the program. This approach is brought ever more to light with the newly constructed field of data science.

As scientist, we build programs to investigate new phenomena or analyze phenomena with which we are already familiar or have a hunch about. All the while, understanding the tentative nature of science, the notion that it exists in a social context, and that it is an ongoing activity that involves uncertainty and subjectivity.

A significant way learning occurs is through play and exploration. When children are engaged in play, they are ostensibly engaged in extending their knowledge base of the world. This concept of exploration as a function of learning is used in some artificial intelligence learning algorithms to help programs discovery interesting patterns or just to learn about their problem space. Similarly, CS students as data scientist have to do the same thing. They have to learn to play with and explore data.

When we are out in the world, we always perceive phenomena in the environment, just like children. Some of these phenomena appear to capture our interest more than others. It is these that we decide to inquire about when we create our hypothesis, conduct our experiments and derive our theories. The idea of **"why"** is one of the essential aspects of scientific inquiry. The whole idea that drives research is our need to learn. In a way, scientists are just like children. One might go further and say that a good scientist should have childlike curiosity.

The transition from exploration to in depth study of the phenomenon is the transition from undirected learning to directed learning. At the heart of scientific inquiry lies three questions (Metz, 2004; Reiser et al., 2001):

1. How do we know what we know?
2. Why do we believe it?
3. And what exactly do we know?

These three questions are the superordinate frames from which scientific inquiry must be understood (Sandoval, 2005). This same three questions guide us in gaining meaningful insights from the computational exploration of data.

How Do We Know What We Know?

This first question deals with explanation driven inquiry, where we attend to how and why things occur as they do. This is what people often think of when they think about the scientific method. It is usually motivated with the paradigm of rule discovery. It is also

what is generally understood by the inquiry cycle which starts with a question we want to answer. We start by formulating a hypothesis or making a prediction, then we design an experiment to confirm or refute our hypothesis. Based on the data that is collected from the experiment, a model is suggested or built, which leads us to apply the newly derived knowledge in context, that leads to another question and the cycle begins again (White and Frederiksen, 1998).

In this view of inquiry, we have to let students know that it is an open-ended exploration instead of the neat little "acquisition of facts" model of science they are used to. That there is no "right" answer, only exploration and knowledge acquisition. On breaking this conditioning, Papert pulls an example from the world of arithmetic and physics education, he states:

> "When computers are used to cure the immediate symptom of poor scores in arithmetic, they reinforce habits of dissociated learning. And these habits which extend into many areas of life are a much more serious problem than weakness in arithmetic. The cure may be worse than the disease. There is an analogous argument about physics. Traditional physics teaching is forced to overemphasize the quantitative by the accidents of a paper-and-pencil technology which favors work that can produce a definite "answer." This is reinforced by a teaching system of using "laboratories" where experiments are done to prove, disprove, and "discover" already known propositions. This makes it very difficult for the student to find a way to constructively bring together intuitions and formal methods."(Papert, 1980b)

In a world where students have been conditioned to believe learning is getting the right answers on a test, or that writing code is mostly for app development, helping them expand their understanding of programming beyond that has to be an end in and of itself.

Why Do We Believe What We Know?

The second questions that lies at the heart of inquiry is being able to defend why we believe as we do. The naive view of science as a settled stable enterprise leads to confusion in the public when scientists argue about the validity of a theory; even though, the history of argumentation is an essential part of what makes science, science (Driver et al., 2000).

Argumentation and reflective inquiry gives us a mechanism for evaluating new knowledge claims. Since science is done in a social context and derives from the subjective view of individuals, this leads to an implicit bias towards "theory-ladeness" (Kuhn et al., 2000). For this reason, observations alone cannot be good enough to evaluate a new claim. This is where argumentation comes in, it is through the defense of ones claim that the warrants are laid bare and can be examined and cross examined by the community until a common consensus is reached with regards to the new knowledge. As was shown in section 4.3, with

race-based policing, based on algorithms that have codified human biases, reflective inquiry provided a mechanism for evaluating the knowledge claims.

Furthermore, through the use of computational data techniques, some researchers are using data gathered through other means to give a more nuanced understanding of the idea of "Black Criminality." The data scientist, Ben Casselman, from Nate Silver's[1] FiveThirtyEight.com states that:

> "As we've written repeatedly, official statistics on police killings are deeply flawed. So the Guardian is building its data set by combining media coverage, reader submissions and open-sourced efforts like Fatal Encounters and Killed by Police, which we've previously found to be reliable" (Casselman, 2015; Five Thirty Eight, 2015)

The data gathered through this approach is open-sourced and freely available through their Github repository. In addition, other data can be submitted to the collection, where the community can verify its authenticity and merge it into the existing datasets. The hope is that a clearer picture can emerge on race-based policing through experiments that are done on these kinds of community developed databases. Through experiments like these, learners can see that CS taken as scientific inquiry can have a profound effect on society.

Argumentation in particular is part and parcel of professional software engineering practice. Through the process of "code-review," software engineers have to justify the approaches they use in writing code. In open-source communities where coding is a collaborative community process, argumentation becomes one of the main processes by which pieces of code are merged back into the main codebase.

For the novice engineer, the process of arguementation can come as a cultural shock. Learning to argue well, learning to give feedback in a supportive manner is a skill that hasn't been folded into standard engineering education, as a result, the practice can easily become a means to abuse and berate junior employees.

What Exactly Do We Know?

The final question at the heart of scientific inquiry is determining the extent of the information that we know, i.e., theory elaboration. Theory elaboration is the process of identifying as many instances of a hypothesis that support a particular theory. This form of scientific inquiry is not often discussed, as it ought. It is from this line of questioning that we build a healthy corpus of data that will eventually completely elaborate a theory.

Now that I have laid out what is meant by scientific inquiry as applied to CS, I want to use the next section to discuss the motivation and creation of the Hip-Hop data module.

[1]Nate Silver is an American Statistician whose blog FiveThirtyEight uses statistical analysis—hard numbers—to tell compelling stories about politics, sports, science, economics and lifestyles.

5.2 Building a Hip-Hop Data Module

My main goal in building this piece of curriculum is to build a bridge between **formal knowledge** and **intuitive understanding**. This is really what data science *is*. As the curriculum designer, my main goal is to create a learning pathway where intuition can flow, and the learner can rely on their own knowledge of the world and their own assumptions, to guide their exploration of embedded knowledge in data.

An approach like this will ultimately lead to a connection between cultural relevance and computing, because each student will have to pull from their pre-existing knowledge, and their own cultural ways of knowing; as a curriculum designer, this is where I can really leverage the diversity of human experience to highlight how computation can refine our specific ways of knowing. To demonstrate my paradigm of data science as the application of formal methods to evaluate intuition, I will now turn my attention to the genesis of the Hip-Hop data module.

Hip-Hop Data Module: Exploration and Play

It all started in 2005 while I was working as a research assistant in a semantic web[2] lab. For many people, when they think of Hip-Hop[3], they don't think artificial intelligence. I, on the other hand do. At the time, I was embedded in a research group that designed and built ontologies, which forced me to adopt an epistemological framework. As a result, I found myself thinking a lot about the development of knowledge. This coincided with a time that I was listening to a lot of rap music, particularly Jay Z.

Everyday I would drive to the lab listening to his music, spending inordinate amounts of time trying to divine the meaning of his lyrics and their implications. There was a particular line in *03 Bonnie and Clyde* that always puzzled me. It went:

> The problem is
> You dudes treat the one that you lovin'
> With the same respect
> That you treat the one that you humpin'

The implication of those lyrics was the person, who was the object of the man's affection, was not the same person with whom the man was sharing his bed. I would often wonder how that line could be parsed and then ran through an intelligent system, which could clearly explicate the implications of the statement.

[2]From Semanticweb.org, The Semantic Web is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites. It has been described in rather different ways: as a utopic vision, as a web of data, or merely as a natural paradigm shift in our daily use of the Web.

[3]Hip-Hop is the culture—encompassing dancing, graffiti, rapping and so on, while rap is the musical aspect of the culture.

Over the years, my love of Jay Z did not diminish, and I was able to see the evolution of thought in his lyrics. The same man, whose lyrics once spoke about being in love with one person and sleeping with another, was now also talking about the miscarriage his wife endured and the birth of his daughter.

As a result of doing semantic web research, I started to see rap lyrics as a computational object to think with; what Papert terms "transitional objects" that gives a bridge from the intuitive to the formal. I experienced something I like to think of as "cognitive cross-over." I started to imagine the kinds of things that could happen in the mind of the scientist when formal computational techniques is brought to bear on a lyrical text analysis of rap music.

Hip-Hop Data Module: Hypothesis

Through my prolonged exposure to rap lyrics, I developed a hunch that MCs[4]—rap poets, have a unique way of knowing that is derived from their expertise as persons who study language, persons who are highly skilled in the art of *Signifying*, and particularly, in the use of novel conceptual metaphors. That was the hypothesis.

What needed to be done to verify this hypothesis was to come up with an experiment that could test it. In order to do that, I first had to explore existing data to see how rappers use language. Upon my initial investigation, I realized that there were no open-sourced rap lyric datasets. If I wanted to do this experiment, I would have to invent such a dataset.

To narrow my focus, I decided to limit the investigation on Jay Z's use of language. I built a webscraper in Python, and scraped together all the lyrics of Jay Z that were in existence at that time. Once I had that initial dataset, I could then start my inquiry into his use of language. These trains of thought were the genesis of the development of this module.

The investigation was deeply embedded in society, had cultural resonance with me—after all I am an avid fan or rap, and was most important "fun!" As with all good curriculum, there was an emotional connection with the work, which enabled the experience of delight in the process of discovery (Newell and Simon, 1976).

Why Rap Analysis is Good for Maximizing Participation

Hip-Hop datasets, unlike an atmospheric datasets from NOAA[5], have a strong emotional resonance with most people because it is *polarizing*. It's one of those evergreen topics for which most people have a strong opinion. They either hate it or they love it. This particular aspect of the genre is important because it doesn't elicit indifference. Whether one hates it, or ones loves it, what matters is, one is engaged and passionate about one's stance on the genre. Its one of those topics that has something for everyone, even for those that hate it.

[4]MC is an abbreviation of Microphone Controller.

[5]NOAA - National Oceanic and Atmospheric Administration, Datasets: https://data.noaa.gov/dataset/international-comprehensive-ocean-atmosphere-data-set-icoads

For the feminist, they can mine it to demonstrate that it has higher rates of misogyny than other genres. For the fans of poetry, they can mine it for its novel use of language. For the cultural critic, they can mine it to get the pulse of what young people are currently into. Most importantly for ethnic minority youth, it can be used to regain a sense of cultural pride through CS education.

1	Data is a part of APCS Principles big idea as can be seen from table 2.3.
2	Its an avenue of engagement for youth in an increasingly global world.
3	It's metaphorically ladened and semantically dense.
4	It provides a great opportunity to bring in issues of gender in general culture, especially misogyny in Hip-Hop.
5	Its an easy way to inject *CS* into *culture*.
6	Could potentially be used as ethno-computing CS curriculum.

Table 5.1: Why a Rap Data Science Module is Particularly Tantalizing

Several qualities contribute to Hip-Hop's effectiveness as a potential computational object to think with. Hip-Hop is part of the natural landscape of many youths of color. It is a field that was pioneered by ethnic minorities, and a field in which ethnic minorities excel. My hope is that through works like this, the association of some ethnic minority youth with education as a vehicle of cultural loss can be regained through positioning CS education as a vehicle for cultural exploration.

Hip-Hop lyrics are metaphorically ladened (Crossley, 2005; Peplow, 2010), they are sufficiently semantically dense and are open to multiple layers of interpretation; allowing wiggle room for each student to explore their own understanding. Table 5.1 lists the reasons why the computational exploration of rap is tantalizing.

Because it resides at the intersection of computer science and culture, the hope is that these kinds of learning contexts will potentially engage students. If as Papert posited and Piaget theorized, that intellectual knowledge is built upon already existing ideas and schema students already possess, then incorporating concepts from mainstream youth culture should help enliven curriculum that is culturally relevant and resonant (Piaget, 1964).

Hip-Hop Data Module: Design Constraints

According to Papert, *powerful* ideas have to be:

Intelligible. Learning should be in "mind-sized" bits, i.e., knowledge should be easy to grasp. For this reason, the data module deals with answering one question, "How does Jay Z use the language of 'basketball' in his lyrics?" That is it. That simple question is what the data module is interested in investigating.

General. The knowledge gained from learning should be applicable over a large domain. Having reflected on the results of the data module experiment, a learner should be able to apply the same methods to a larger corpus of rap lyrics to determine whether the finding in Jay Z's lyrics can be generalized over an entire rap corpus. More importantly, the techniques *learned* in conducting the experiment should be easily transfered over to textual analysis of other kinds.

Personal. Learning should not lead down the road to disassociation has we have seen in environments that pose a threat to student's learning. By grounding the data module in rap lyrics, I seek to follow this guiding principle by bringing in a cultural genre that most youth a familiar with.

The fully elaborated lab can be found in appendix A of this work. I follow the learning with data computation framework of Dasgupta to guide the creation of this learning module, I extend the computational framework to include visualizing data (Dasgupta, 2012). Furthermore, I model elements of the module to correlate the data science workflow—figure 5.2, that is usually used in industry as can be seen in table 5.2.

There are some limitations to the workflow posited in figure 5.2, the model assumes you never need to go back to the beginning, realizing that the data you have does not allow you to answer the question you want. In a robust model, there should be an arrow back to "acquire" more "data." This model also does not allow for the experiment to be a failure when the data does not reveal any meaningful insights.

Hip-Hop Data Module: Curriculum

Objectives

The student will be able to

- Come up with an question they will like to investigate through the exploration of data
- Write an algorithm to query a dataset
- Build a visualization of the results of a dataset query

Outline of Lesson

- Introduction to Data in Python
  - Welcome to computational lyrical analysis using Python
- Introduction to NLTK package
  - Getting familiar with the Natural Language Toolkit

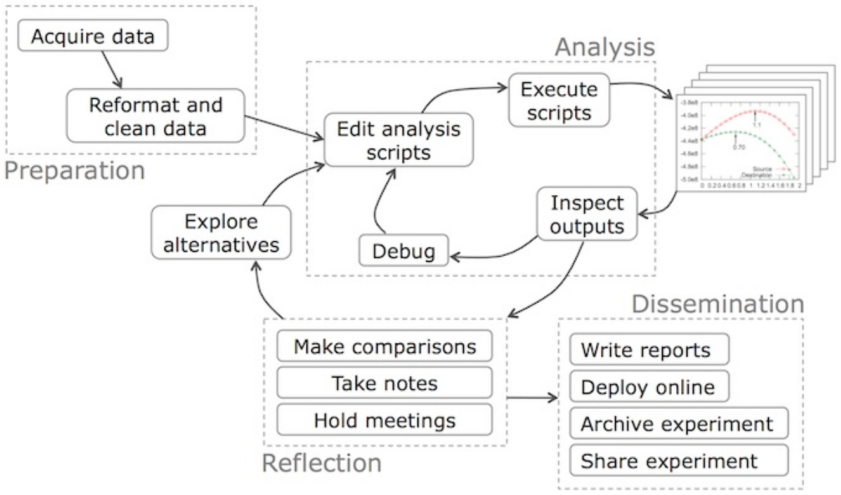


Figure 5.1: Data Science WorkFlow diagram by (Guo, 2013)

Tasks	Sub Tasks	Realization in Hip-Hop Data Module
Preparation	Acquire data	A.3, The learner is tasked with building a corpus.
	Reformat and clean data	A.4, A.5 The learner is exposed to the idea of stemming data to make conceptual investigations more scalable.
Analysis	Create analysis scripts	A.5 In this part of the lab, the focus is on formalizing intuition by first understanding the relationship between the layers of abstraction that store the data.
	Inspect output	A.5 Once the query is written, here the lab focused on the nuance interpretation of the output based on inspection of data plots.
Reflection	Make comparisons	A.5 Through the creation of a concordance, the learner is ushered into a mindset of reflecting on the output of the data.
	Take notes	Outside of the scope of the lab.
Dissemination	Write report	Outside of the scope of the lab.
	Share experiment	

Table 5.2: Mapping Data Science Work Flow to Hip-Hop Data Module

- Creating a Jay Z Corpus
  - Corpus Reader
- Frequency Analysis
  - Determine the number of unique words used within an artist's first 35,000 lyrics
- Data crunching versus gaining insight? When Data Science runs amok
  - Lets investigate basketball concepts in the corpora

Big Ideas

- "Computational thinking" is the metacognitive process that attunes to the way we solve problems through the use of computational methods. It can apply to both machine dependent and machine independent processes.
- Python is a language suitable for scientific computing.
- After learning one language, the few languages you learn will be much easier. What you're learning is "how to learn". In the case of CS10, "The Beauty and Joy of Computing,"you started your computational problem solving journey using Snap*!*, now we have transitioned to Python. While Snap*!* was a graphical block-based language, and Python is text-based, what is of most importance is that they are both languages that can be used to realize computational solutions.
- Python is well suited for text analysis because of its powerful nltk (Natural Language processing) toolkit.

Activities

- Investigate an idea in literary corpora.
- Learn the basics of Python text analysis and data manipulation, including file I/O.

Outcome

- Gain familiarity with elementary techniques of natural language processing.

Discussion of Data Module: Empathy

To a certain extent, one could argue that the data module that I designed perpetuates the same dissociated learning paradigm that Papert rails against. However, what makes the approach I have taken different is that the series of exercises are an opportunity for learners to see how this kind of thinking is done. It isn't merely a data science tutorial, but more an experiential peek into the mind of a data scientist as they bring formal methods to bear on their intuition.

Lyrical exploration can be used to create a cognitive social connection. You can project yourself as the rap poet composing the lyric, carefully playing with words so they have double meanings based on context. You can *intuit* your way into figuring out how to find these clever gems. Most importantly it forces you, the data scientist, to take the point of view of another person, and try to experience the world through their words; automatically taking you away from being "egocentric" to being "altruistic."

Discussion of Data Module: Cultural Resonance

This data module is also an exercise in helping students get exposure to different ways of knowing by investigating language use. This is particularly important in CS because its an approach that allows for a non-confrontational way to talk about diversity—in this case, diversity of thought.

Picking up the thread of thought from chapters 1.4 and 2.2, I had outlined the following concern.

> "In the CS context, computational thinking presents a tantalizing opportunity for culturally responsive pedagogy. Because of its recent ascendancy, there is very little research both empirical and theoretical on how culturally responsive pedagogy that supports computational thinking can be done." - Chapter 1.4

This data module demonstrates how culturally responsive pedagogy can be realized in a computational thinking based curriculum. This module specifically positions computation as a means of investigating culture.

I want to emphasize that I do not claim that there is an "African American," or "Asian American" way to teach CS, but more importantly, that *even* CS with its layers and layers of abstraction and mathematical complexity, can be used as an object of cultural inquiry.

Chapter 6

Findings

"In social species, individuals are also members of groups. An important part of their cognitive activity is directed toward other members of the group with whom they cooperate and compete."

— (Sperber and Hirschfeld, 1999)

What is science but a story of cause meets effects. I begin to tell the story of what I have found in my study of the experience of undergraduate students in introductory computer science at UC Berkeley, by first giving a report of the statistically significant findings in the quantitative data analysis, followed by a brief discussion of the major themes that emerged in the qualitative analysis.

6.1 Quantitative Data Analysis

I start my investigation around the experiences of students in CS10 and CS61A as quantified through their responses to survey instruments, which were disseminated and coded to facilitate analysis. The instruments can be found in appendix E. The code table for the instrument can be seen in table 6.1.

Table 6.2 shows the results of statistical test for significance—Using Mann Whitney Test—between the experience of students in CS10 and CS61A. Only survey instruments for whom p values were less than 0.05 where included in the table.

As a result of the normalization of the data, the numbers denoting the responses of the students do not quite add up to 100. For the most part, they add up to 100 ± 2, because of precision loss. An elaborated analysis process for the quantitative data collected in this study can be found in appendix B.

Table 6.1: Code Table for Survey Data

Survey Instrument	Definition
atcs_1	I like to use Computer Science to solve problems.
atcs_2	I can learn to understand computing concepts.
atcs_3	I can achieve good grades (C or better) in computing courses.
atcs_4	I do not like using computer science to solve problems.
atcs_5	I am confident I can solve problems by using computation.
atcs_6	The challenge of solving problems using computer science appeals to me.
atcs_7	I am comfortable with learning computing concepts.
atcs_8	I am confident about my abilities with regards to computer science.
atcs_9	I do think I can learn to understand computing concepts.
atct_2	I have good research skills.
atct_3	I am good at using online search tools.
atct_5	I know how to write computer programs.
atct_6	I am good at building things.
atct_7	I am good at ignoring irrelevant details to solve a problem.
atct_8	I know how to write a computer program to solve a problem.
atcsgender_1	Women are less capable of success in CS than men.
atcsgender_2	Women are smarter than men.
blg_1	In this class, I feel I belong.
blg_2	In this class, I feel awkward and out of place.
blg_3	In this class, I feel like my ideas count.
blg_4	In this class, I feel like I matter.
clet_2	I think about the ethical, legal, and social implications of computing.
cltrcmp_2	I have good cultural competence, or the ability to interact effectively with people from diverse backgrounds.
snap_python	I was able to see how computational thinking carries over from Snap*!* to Python.
hiphop_d1	The hip-hop data module made me see how computation can enhance culture.
hiphop_d2	Lyrical analysis gave me a new way to think about computation.
song_ct	I was able to build on my pre-existing knowledge about songs to further understand computation.

Table 6.2: Survey Finding with Statistical Significance

Survey Instrument	P Value	Significance by Stars
atcs_1	0.00000	****
atcs_2	0.00011	****
atcs_3	0.00424	**
atcs_4	0.00000	****
atcs_5	0.00035	***
atcs_6	0.00001	****
atcs_7	0.00000	****
atcs_8	0.01949	**
atcs_9	0.00004	****
atct_5	0.00000	****
atct_8	0.00000	****
atcsgender_2	0.00913	**
blg_3	0.00001	****
blg_4	0.00000	****
clet_2	0.00000	****

Table 6.3: Survey Finding with Statistical Significance [Data Disaggregated]

Survey Instrument	Sample Group	P Value	Significance by Stars	P Value	Significance by Stars
		Prior CS		**NO Prior CS**	
atcs_1	Class	0.02506	*		
atcs_2	Class	0.00448	**		
	Male	0.00883	**		
atcs_3	Class	0.01555	**		
	Class			0.00895	**
	Female			0.00030	***
	Male	0.02903	*		
atcs_4	Class	0.04441	*		
	Class			0.00341	**
	Male			0.00573	**
atcs_6	Class	0.00638	**		
	Male	0.00852	**		
atcs_7	Class	0.00064	***		
	Male	0.02253	*		
atcs_9	Class	0.01197	**		
atcsgender_1	Female			0.00214	**
atcsgender_2	Class	0.01138	**		
atct_2	Female	0.02310	*		
atct_3	Class			0.00911	**
	Female			0.00258	**
atct_5	Class	0.00000	****		
	Class			0.00627	**
	Female	0.00024	***		
	Male	0.00001	****		
atct_6	Class	0.02987	*		

Survey Finding with Statistical Significance [Data Disaggregated] (*continued*)

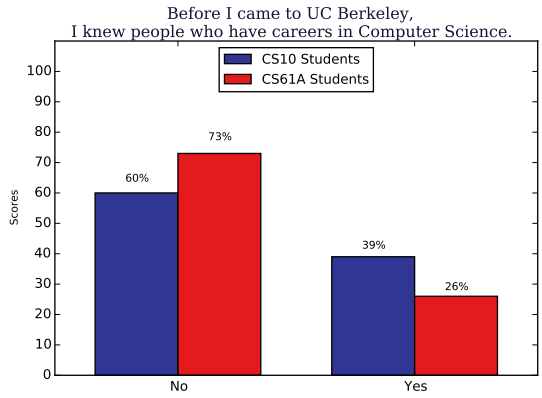
Survey Instrument	Sample Group	P Value	Significance by Stars	P Value	Significance by Stars
		Prior CS		**NO Prior CS**	
atct_7	Class			0.03963	*
	Female			0.04346	*
atct_8	Class	0.00002	****		
	Class			0.02489	*
	Male	0.00268	**		
blg_1	Class			0.00023	***
	Female			0.00104	***
	Male			0.00755	**
blg_2	Class			0.00736	**
	Female			0.01275	**
blg_3	Class	0.00393	**		
	Class			0.00001	****
	Female	0.00024	***		
	Female			0.00028	***
	Male			0.00050	***
blg_4	Class	0.00716	**		
	Class			0.00000	****
	Female	0.00121	***		
	Female			0.00013	****
	Male			0.00002	****
clet_2	Class	0.00246	**		
	Class			0.00000	****
	Female	0.00490	**		
	Female			0.00202	**
	Male			0.00013	****
cltrcmp_2	Female	0.00804	**		

Findings:
On Mentorship

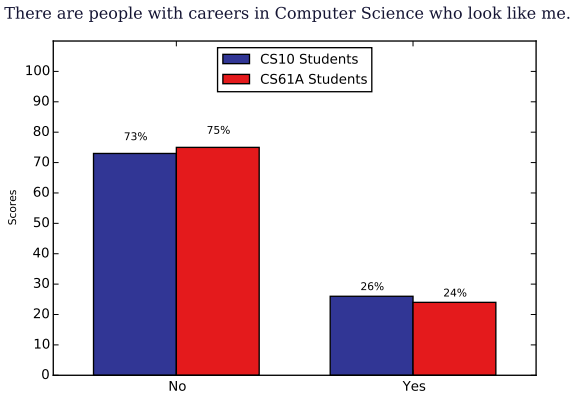
6.2 Findings

On the Role of Mentorship

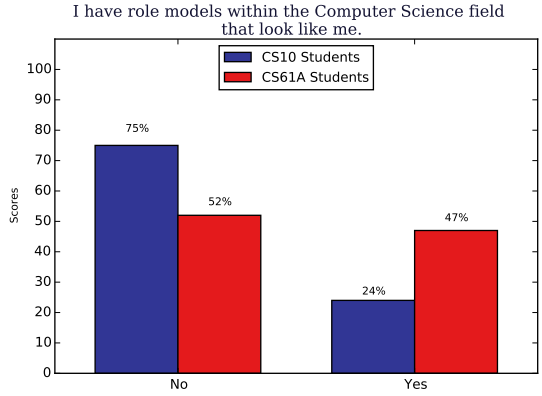
I wanted to see what role mentorship played in the experience of learners. Especially, in the experience of historically underrepresented groups. One unexpected thing that I found is the following: when asked this question, "there are people with careers in CS that look like me," 81% of CS10 male students and 71% of CS61A male student answered "No" to that question—figure 6.2d. Female students seemed to know more career professionals in CS, and in particular, female students in CS10 as one can observe from 6.2c.



(a) mtr_1

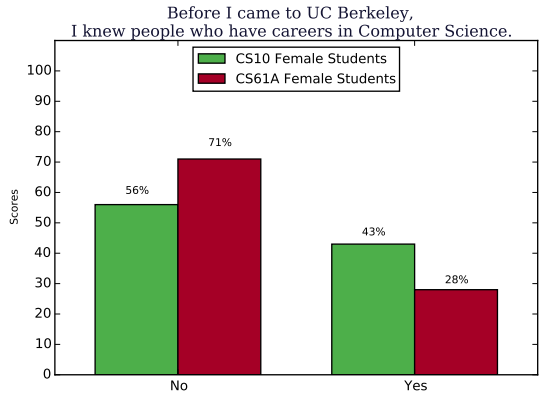


(b) mtr_2

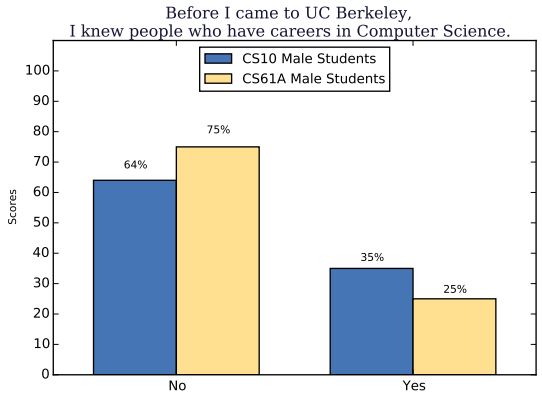


(c) mtr_3

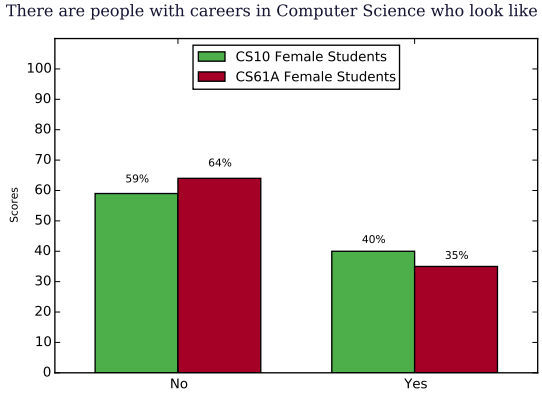
Figure 6.1: Students' Responses with Regards to Mentorship.



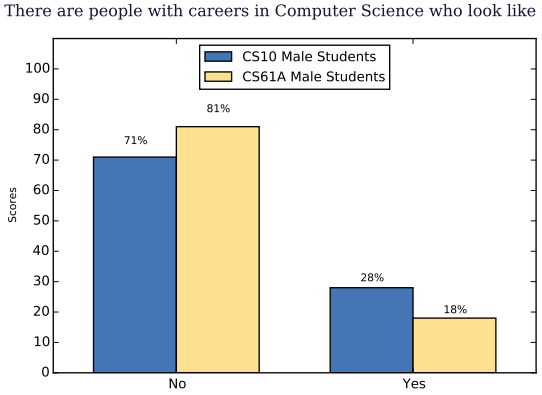
(a) Female Respondents: mtr_1



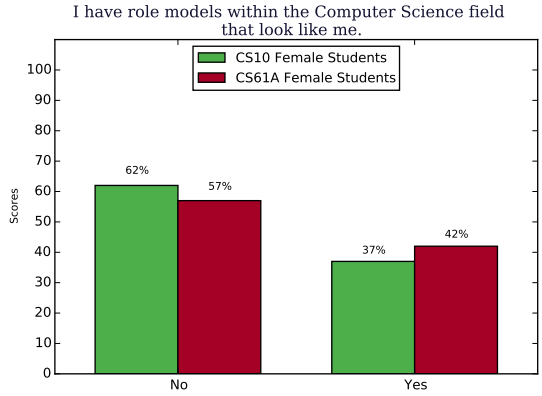
(b) Male Respondents: mtr_1



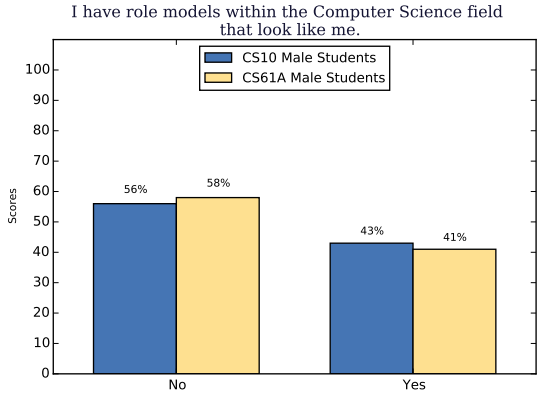
(c) Female Respondents: mtr_2



(d) Male Respondents: mtr_2



(e) Female Respondents: mtr_3



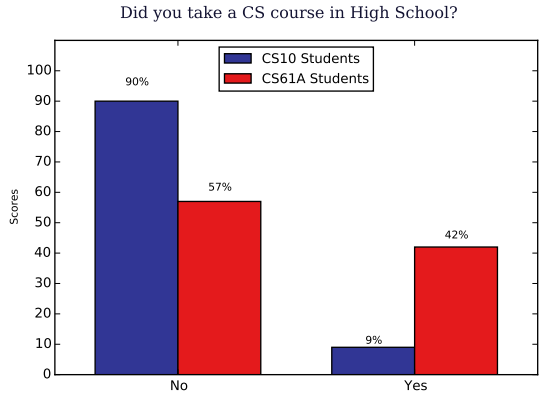
(f) Male Respondents: mtr_3

Figure 6.2: Students' Responses with Regards to Mentorship.

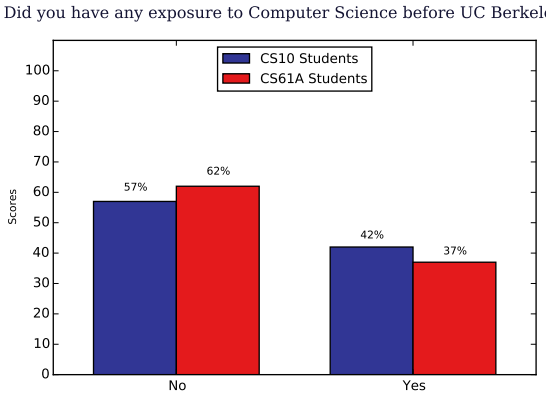
Findings:
The Role of Family in Choosing CS

On the Role of Family in CS Choice

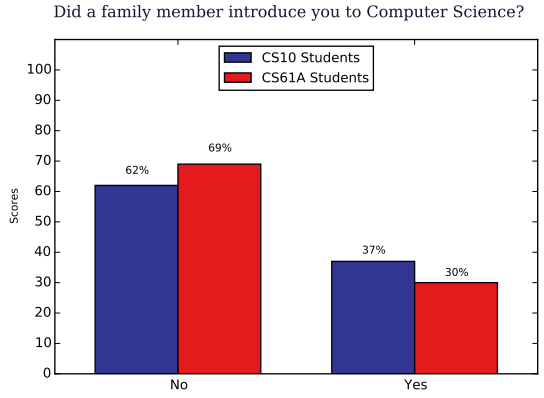
When asked about exposure to CS before arrival on campus, of all the demographics, female students in CS61A have the highest exposure to the field, at 50%—figure 6.4a. Furthermore, 45% of those same women have a close family member who is a computer scientist—figure 6.4e.



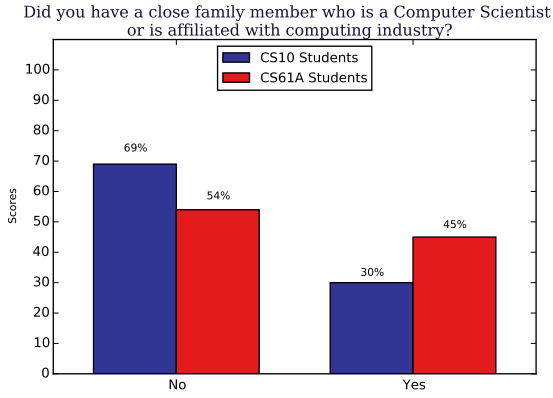
(a) prcs_1



(b) prcs_2

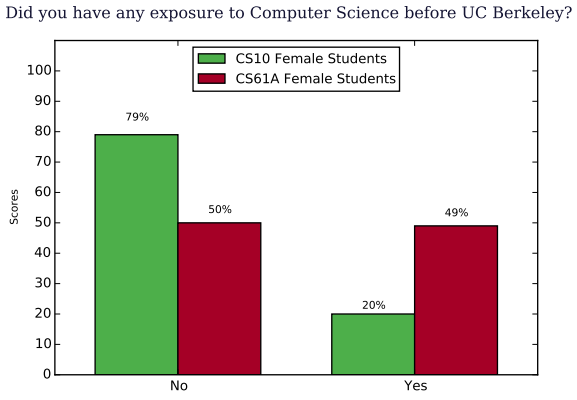


(a) prcs_3

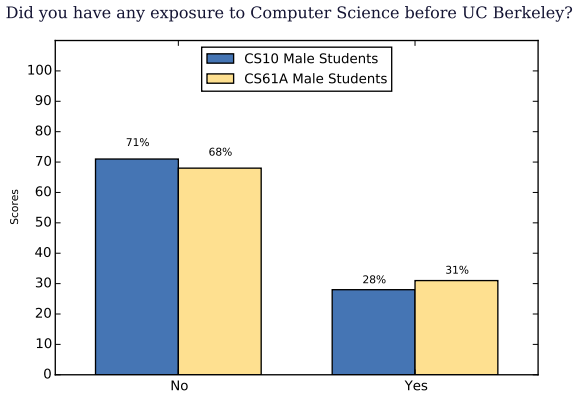


(b) prcs_4

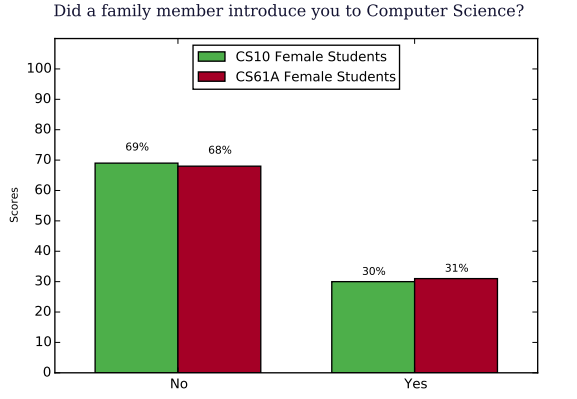
Figure 6.3: Students' Responses with Regards to Pre-Collegiate CS Exposure.



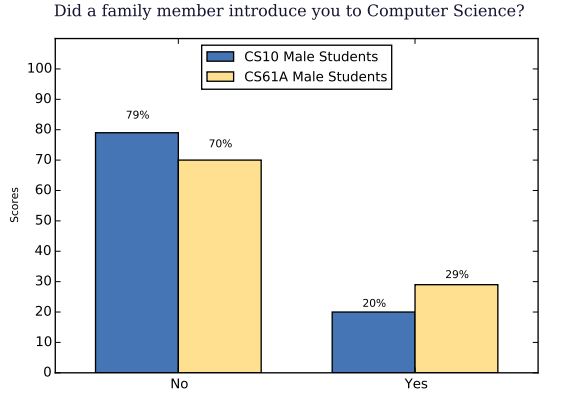
(a) Female Respondents: prcs_2



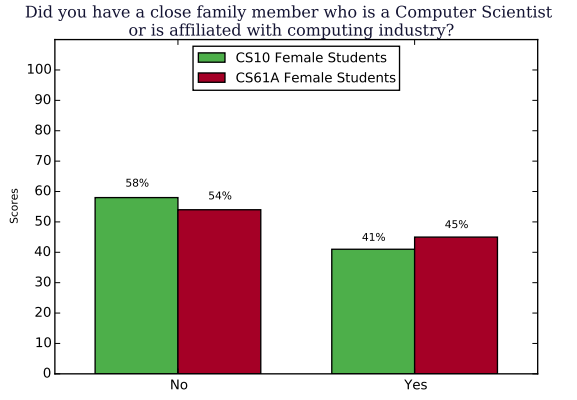
(b) Male Respondents: prcs_2



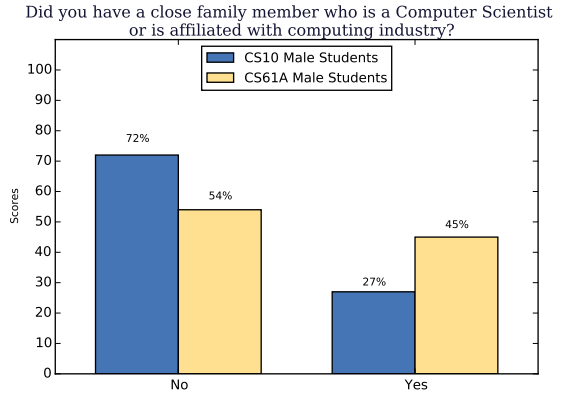
(c) Female Respondents: prcs_3



(d) Male Respondents: prcs_3



(e) Female Respondents: prcs_4



(f) Male Respondents: prcs_4

Figure 6.4: Students' Responses with Regards to CS Exposure.

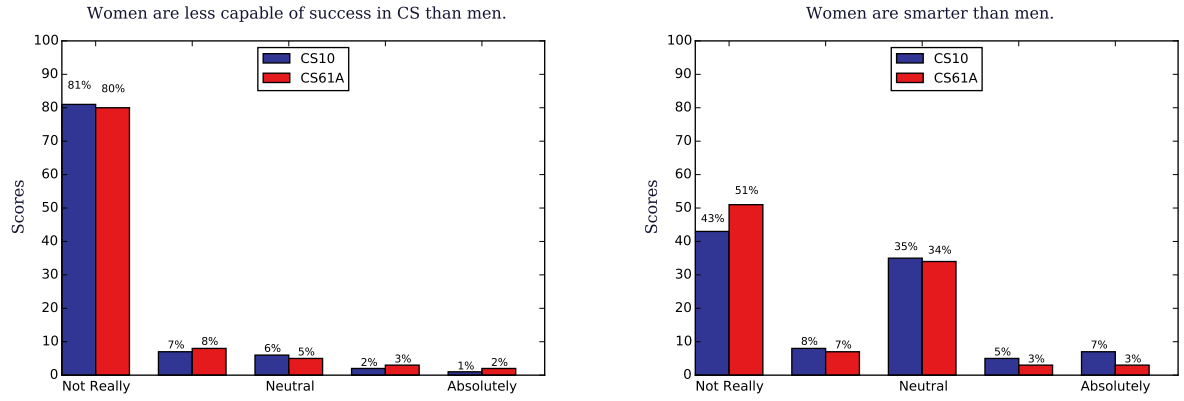
Findings:
On Gendered Ideas of Intelligence

On Gendered Ideas of Intelligence

Just to be certain that the reason why women were not advancing in the computer science pipeline at a rate commiserate to their representation at the college level, was not female students belief about "innate female CS intellectual inferiority," students were asked to provide data on their understanding of the relationship between gender and intelligence. Two survey items were created, and coded, "atcsgender_1:Women are less capable of success in CS than men", and "atcsgender_2:Women are smarter than men."

The result confirmed, with statistical significance that female students strongly reject the notion of the myth of male superior intellectual ability being the basis of their computer science success, as evidenced in figure 6.5a. Even more encouraging is the slight increase in women's self-reported belief about their capability for CS in comparison to their male counterparts, in the responses of female students' who have had prior exposure to CS in both CS10 and CS61A, as can be see in figure 6.6c and 6.6d. For women without prior exposure to CS, their was an effect with ($p = 0.00214$).

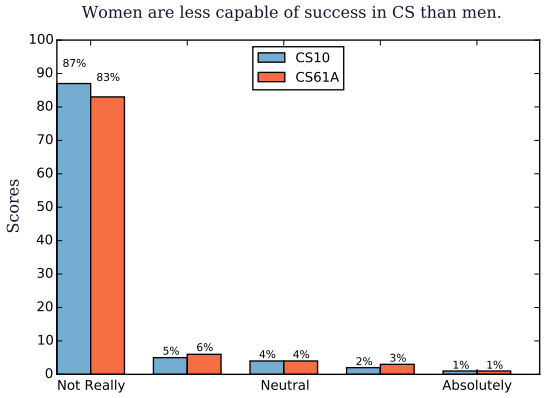
It is important to notice that when asked about relative intelligence based on gender, around 30%+ students in both classes remained neutral, as can be seen from figures 6.5b, 6.7. The male students in this study showed no statistical significant responses around gendered notions of intelligence.



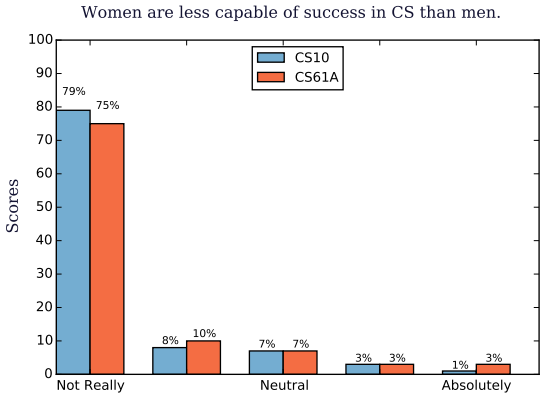
(a) atcsgender_1

(b) atcsgender_2

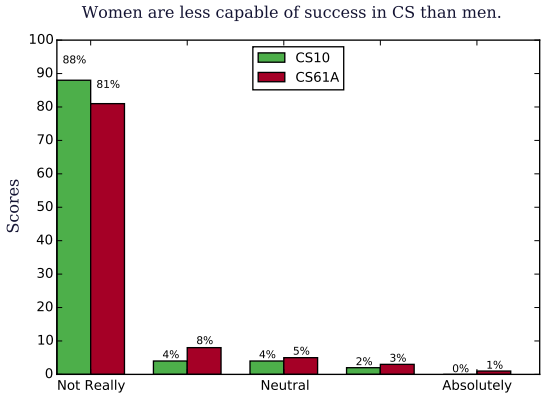
Figure 6.5: Students' Responses with Regards to Gendered Ideas.



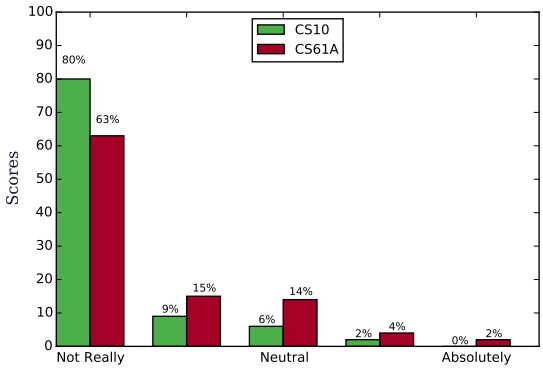
(a) Respondents with **Prior CS**: atcsgender_1



(b) Respondents with **No Prior CS**: atcsgender_1

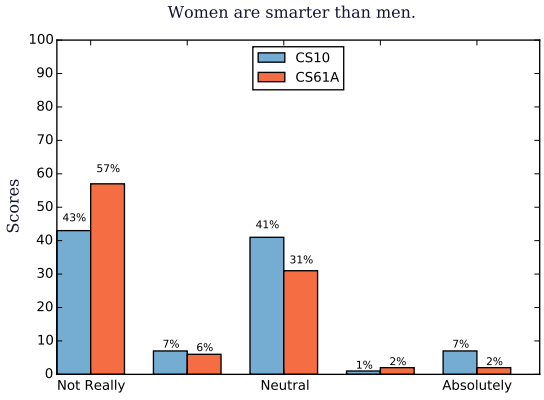


(c) **Female** Respondents with **Prior CS**: atcsgender_1

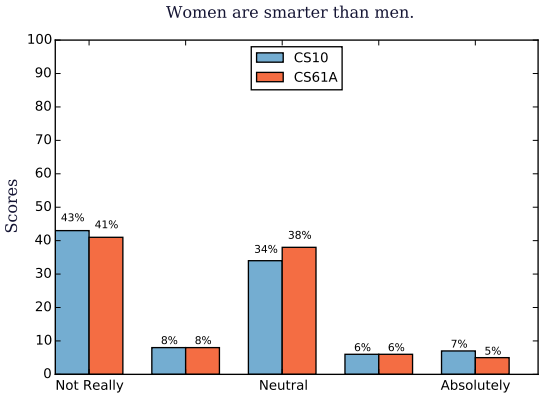


(d) **Female** Respondents with **No Prior CS**: atcsgender_1

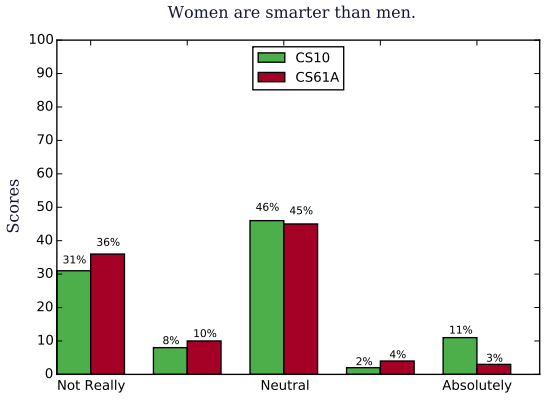
Figure 6.6: Students' Responses with Regards to Gendered Notion of CS Success.



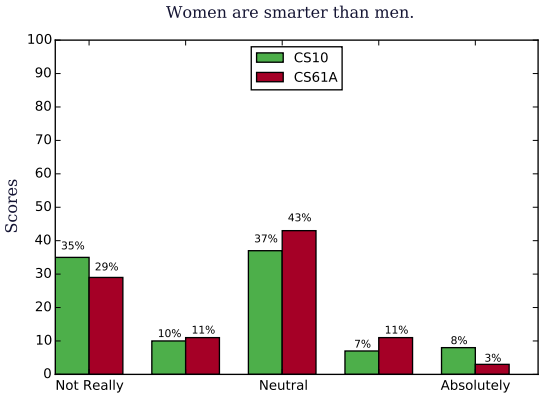
(a) Respondents with **Prior CS**: atcsgender_2



(b) Respondents with **No Prior CS**: atcsgender_2



(c) **Female** Respondents with **Prior CS**: atcsgender_2



(d) **Female** Respondents with **No Prior CS**: atcsgender_2

Figure 6.7: Students' Responses with Regards to Gendered Notion of Intelligence.

Findings:
On Social Implications of Computing

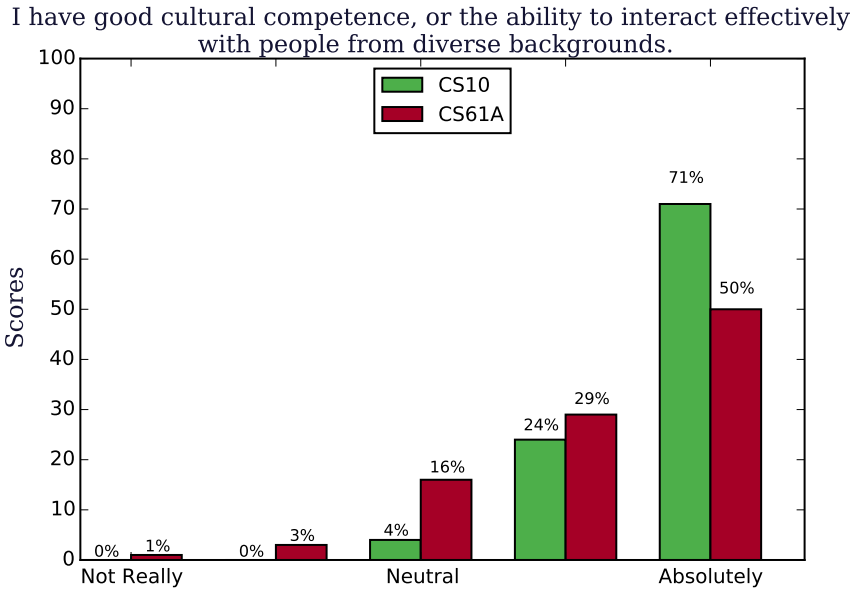
On Social Implications of Computing

To find out if CS10 learners are indeed engaging the world critically, four survey instruments were developed and coded as follows:

clet_1: I work well in teams.
clet_2: I think about the ethical, legal, and social implications of computing.
cltrcmp_1: I am comfortable interacting with peers from different backgrounds than my own (based on race, sexuality, income, and so on.)
cltrcmp_2: I have good cultural competence, or the ability to interact effectively with people from diverse backgrounds.

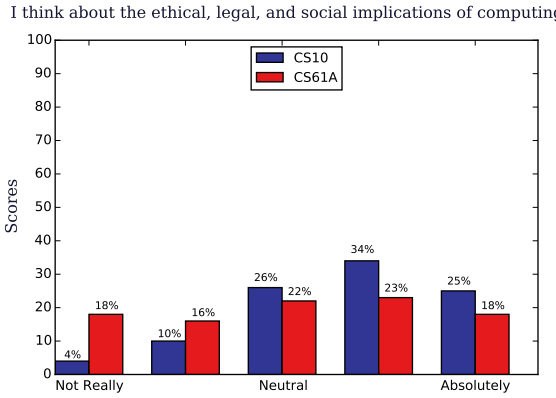
The impacts of the social implications of computing thread in CS10 can be seen from the data plotted in figure 6.9. We observe a statistically significant effect with ($p = 0$), with respect to their cohorts in CS61A, along both genders.

When asked about their cultural competency—cltrcmp_2, I found a statistically significant effect with ($p = 0.00804$), especially with female student with prior CS exposure, as can be seen from figure 6.8a.

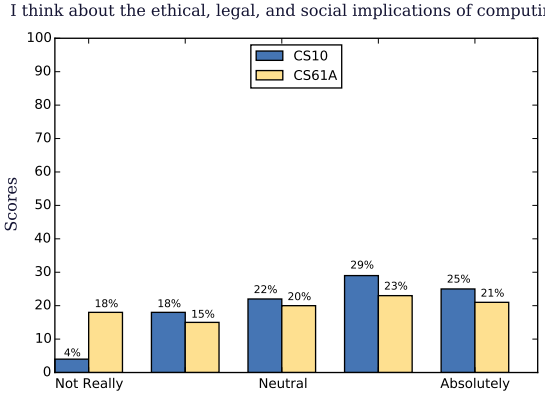


(a) cltrcmp_2

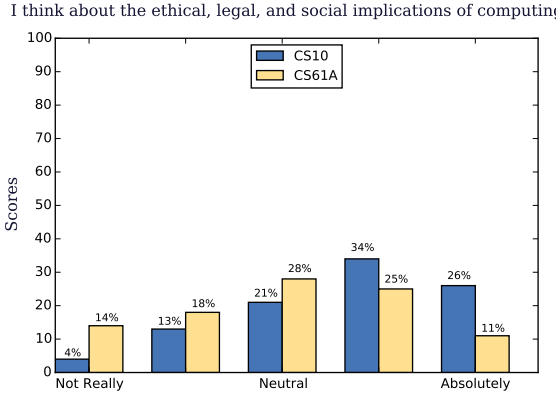
Figure 6.8: **Female** Respondents with **Prior CS**: cltrcmp_2.



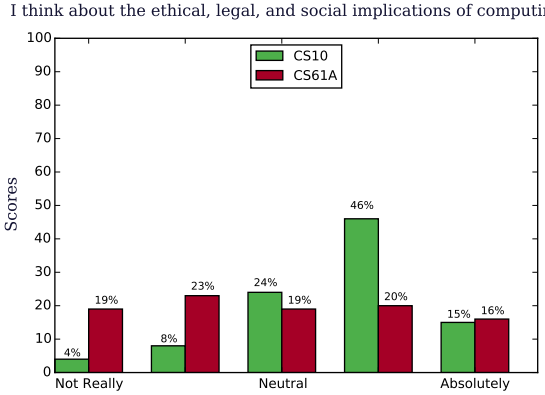
(a) clet_2



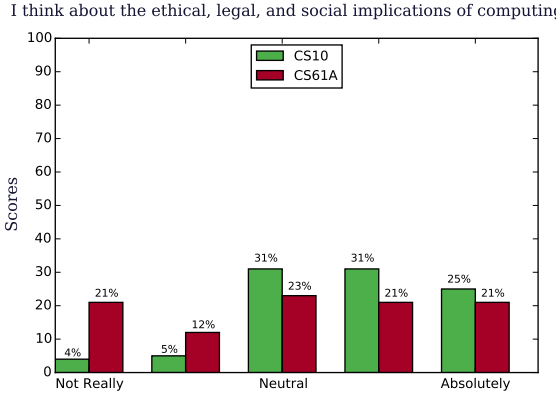
(b) **Male** Respondents with **Prior CS**: clet_2



(c) **Male** Respondents with **No Prior CS**: clet_2



(d) **Female** Respondents with **Prior CS**: clet_2



(e) **Female** Respondents with **No Prior CS**: clet_2

Figure 6.9: Students' Responses to Social Implications of Computing.

Findings:
On Belonging

On Belonging

To investigate students' perception of CS fitness, as well as their sense of efficacy in the CS classroom, four instruments were designed and coded as follows:

blg_1: In this class, I feel I belong.
blg_2: In this class, I feel awkward and out of place.
blg_3: In this class, I feel like my ideas count.
blg_4: In this class, I feel like I matter.

Based on the data gathered from the survey, students generally had a stronger, statistically significant experience of belonging in CS10 as compared to CS61A, as can be seen in figure 6.10. Nevertheless, it is important to notice that only around 50% of female learners had a positive sense of belonging ($p = 0.00104$) as can be seen from figure 6.10b.

When focusing on ideas, only about 50% of female learners felt their ideas were important in the class ($p = 0.00028$), as can be seen in figure 6.11f.

When asked about their sense of feeling important in the class, we observe a statistically significant ($p = 0.0000$) difference between the experience of learners in CS10 versus their counterparts in CS61A, figure 6.10f.

Narrowing our analysis to the experience of learners without prior exposure to CS, we see that for CS61A, similar proportions of learners of both gender feel like they don't matter, i.e., 35% ±5. In contrast, only 10% of boys ($p = 0.00002$) felt unimportant versus 17% of girls ($p = 0.00013$) in CS10. Important to note is that when we look at female learners with prior CS exposure, the proportion drops to 8% ($p = 0.00121$).

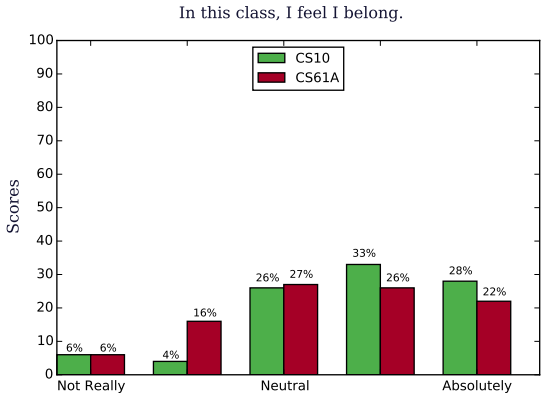
Effect of CS10 on CS61A Experience: Belonging

First, lets take a gendered look at learners self-reported belonging. From figure 6.13a, we observe that 51% of female learners have positive sense of belonging versus 61% of male learners.

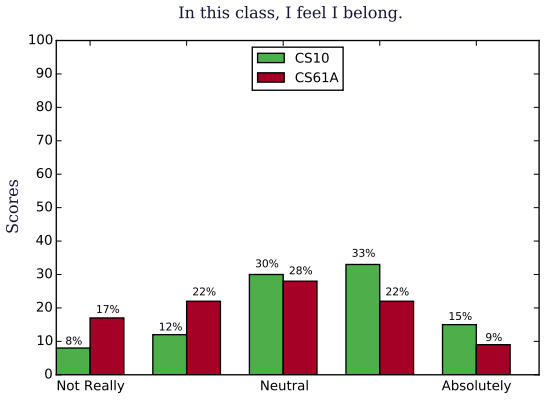
To see what effect participation in CS10 has on students' sense of belonging in CS61A, I created a data subset of CS61A students who had previously taken CS10. This new subset had a sample size N=40, with 18 female subjects and 22 male subjects.

Looking to see if the experience of CS10 increased female students sense of belonging when they got to CS61A, with respect with their male counterparts, we observe a statistically significant ($p = 0.02006$) difference. Reducing the dimension of the likert scale responses by combining the two dimensions on either side of "neutral." From figure 6.13d we observe that *now* only 27% of female students feel a sense of belonging.

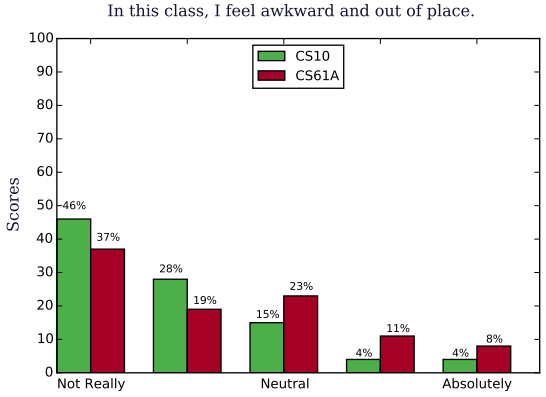
Comparing female students with zero CS experience against those that took CS10 previously, we observe no statistically significant effect. You can see the results in figure 6.13e.



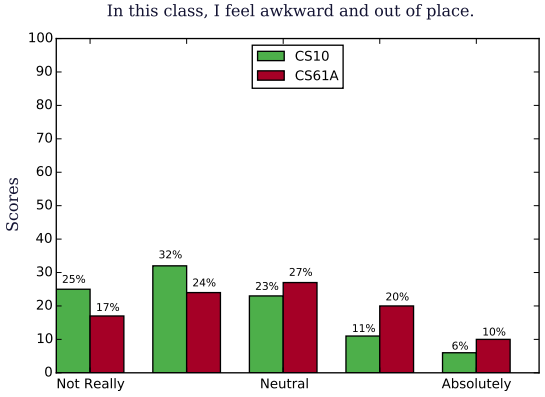
(a) **Female** Respondents with **Prior CS**: blg_1



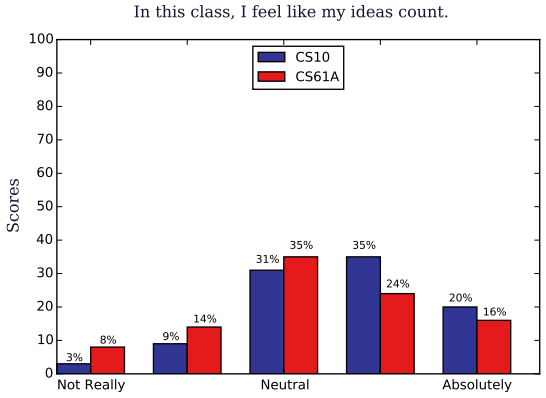
(b) **Female** Respondents with **No Prior CS**: blg_1



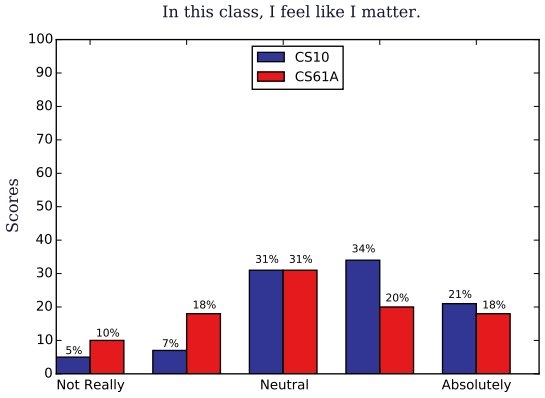
(c) **Female** Respondents with **Prior CS**: blg_2



(d) **Female** Respondents with **No Prior CS**: blg_2

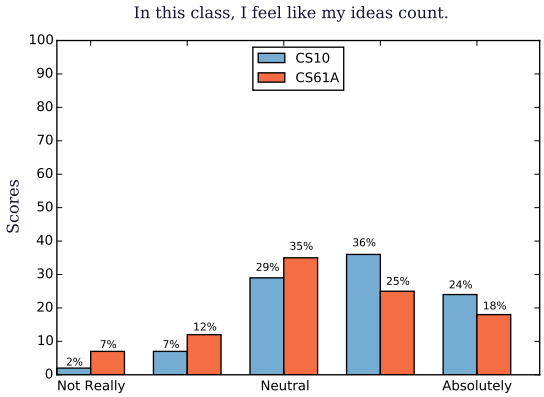


(e) blg_3

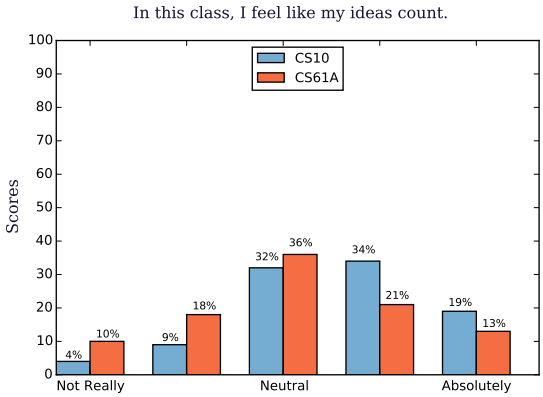


(f) blg_4

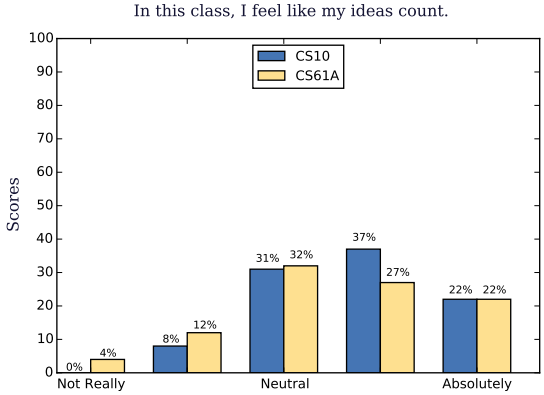
Figure 6.10: Students' Responses with Regards to CS Belonging.



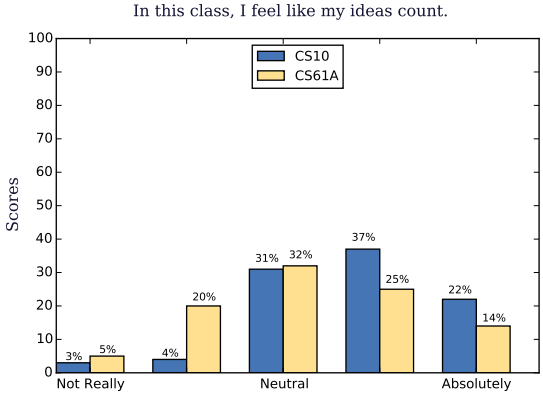
(a) Respondents with **Prior CS**: blg_3



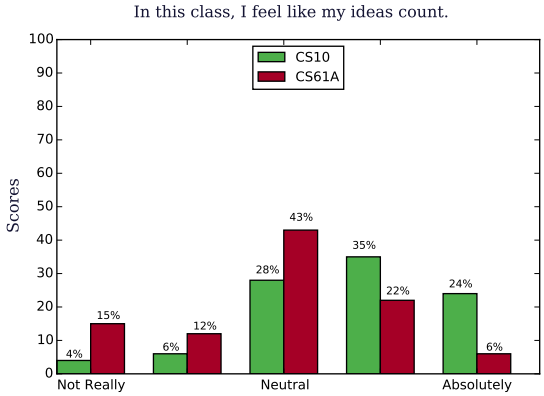
(b) Respondents with **No Prior CS**: blg_3



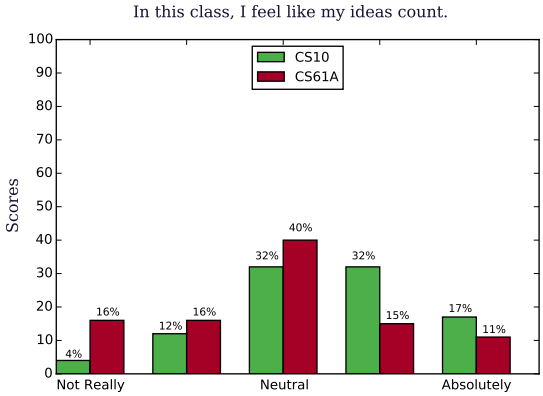
(c) **Male** Respondents with **Prior CS**: blg_3



(d) **Male** Respondents with **No Prior CS**: blg_3

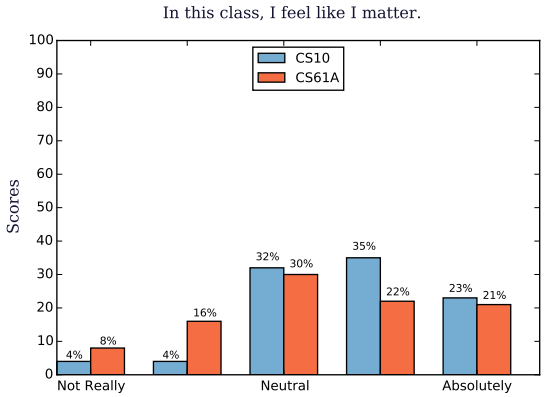


(e) **Female** Respondents with **Prior CS**: blg_3

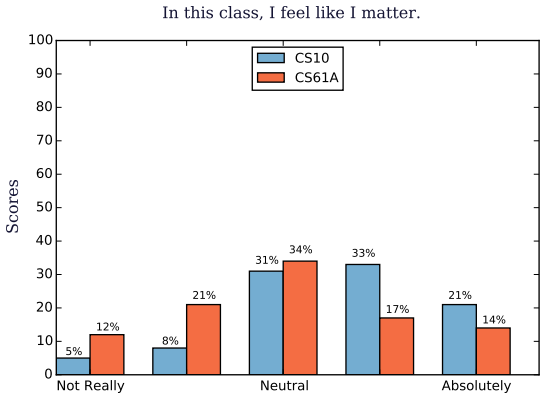


(f) **Female** Respondents with **No Prior CS**: blg_3

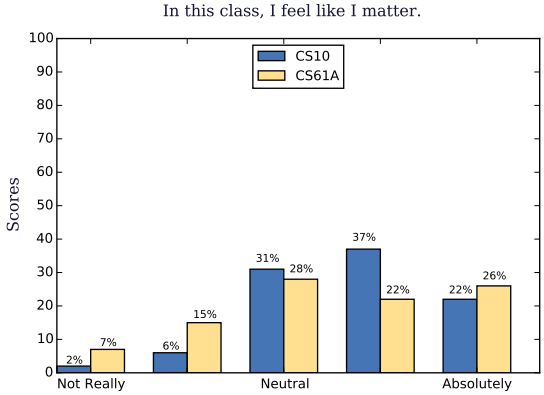
Figure 6.11: Students' Responses , (Data Disaggregated).



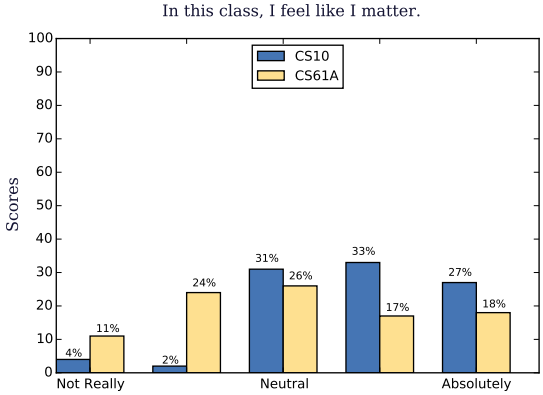
(a) Respondents with **Prior CS**: blg_4



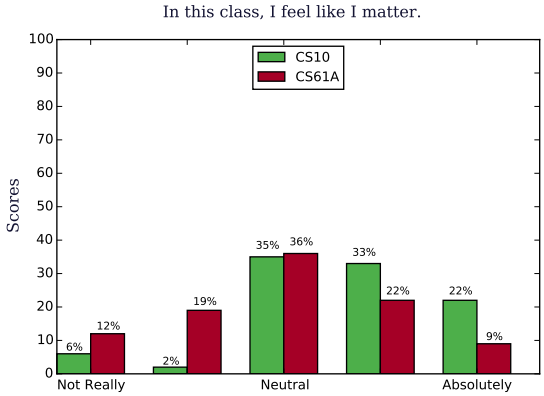
(b) Respondents with **No Prior CS**: blg_4



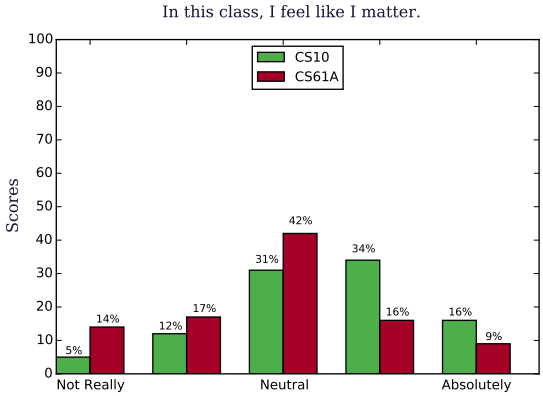
(c) **Male** Respondents with **Prior CS**: blg_4



(d) **Male** Respondents with **No Prior CS**: blg_4

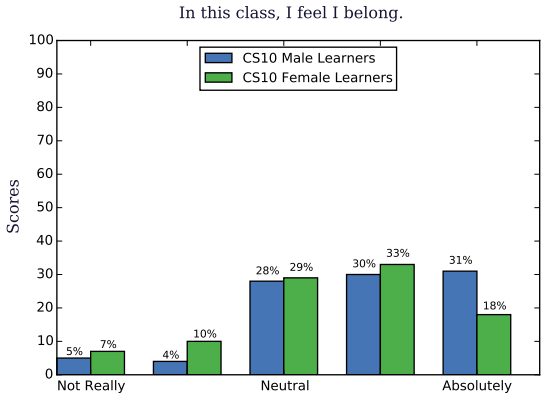


(e) **Female** Respondents with **Prior CS**: blg_4

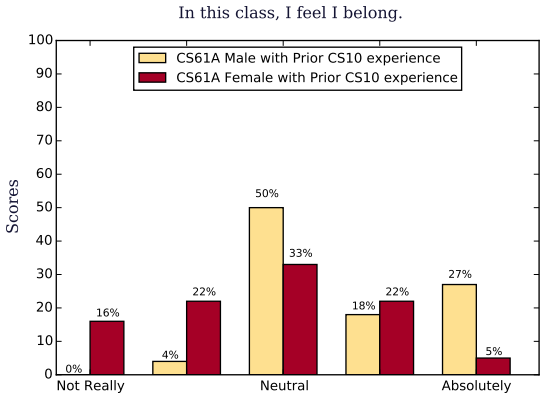


(f) **Female** Respondents with **No Prior CS**: blg_4

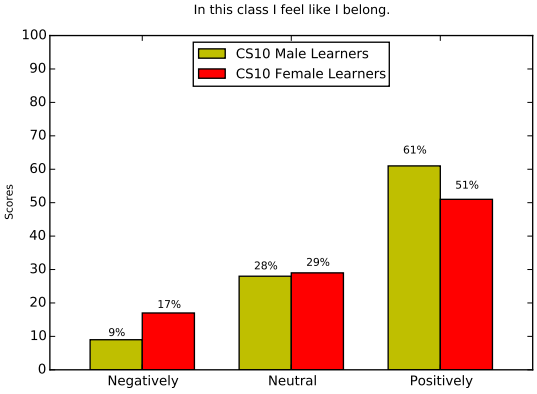
Figure 6.12: Students' Responses , (Data Disaggregated).



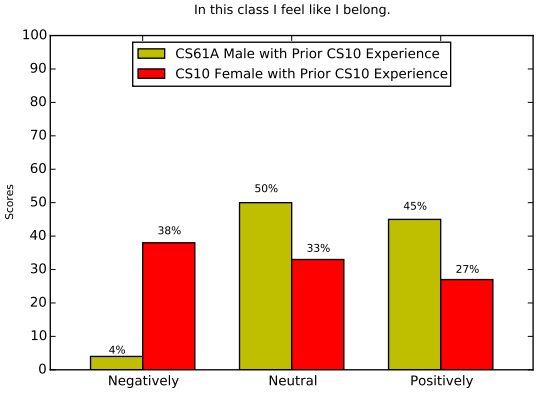
(a) CS10 Learners: blg_1



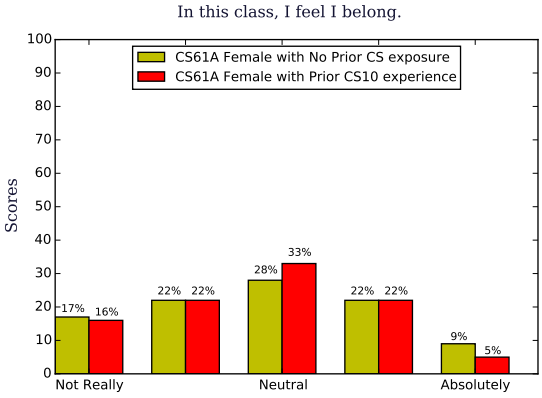
(b) CS61A Learners with **Prior** CS10: blg_1



(c) CS10 Learners: blg_1



(d) CS61A Learners with **Prior** CS10: blg_1



(e) **blg_1**

Figure 6.13: Effects of CS10 on CS61A Students' Self-Reported Sense of Belonging.

Findings:
On Computer Science Efficacy

On Computer Science Efficacy

Along the dimension measuring students' computer science efficacy—coded "atcs_[x]", we see from table 6.2, there is a compelling statistical significance in the experience of students in CS10 as compared to students in CS61A. Almost on every instrument along this dimension, CS61A students have a higher self-reported score than their peers in CS10, as we can see from the data in figure 6.14. The only instrument on which the scores are somewhat commensurate is when asked the question "I am confident about my abilities with regards to computer science," as can be seen in figure 6.14e.

To get a finer grain view of what is going on, I disaggregate the data along gender and prior exposure to CS. Once that is done, from results displayed in figure 6.15, and based on the statistical significance of the disaggregated survey findings as is displayed in table 6.3, we can see that the survey instrument coded as atcs_2, has a slight statistical significance in the experience of student with prior CS exposure, and in particular the experience of the male students in that cohort. If we look at the data plotted in figure 6.15d, we see that male students in CS61A, skew more towards absolute certainty towards their ability to understand CS concepts than their CS10 counterparts. The female student's registered no statistical significant experience.

On Belief Around CS Achievement

When assessing self efficacy around students' belief in excelling in CS, as measured by the instrument "atcs_3: I can achieve good grades (C or better) in computing courses," we find interesting results. There is a clear statistical difference in the beliefs of students with prior exposure to CS with ($p = 0.00424$) than those without.

Looking at figure 6.16b, we can see that 81%[1] of students in CS61A with prior exposure to CS believe they can get good grades in CS as opposed to 77% of CS10 students in the same cohort with ($p = 0.01555$).

However, when you look at the experience of students without prior CS exposure, the CS10 student's in that cohort have a higher sense of belief in their ability to achieve good grades in CS than their CS61A counterparts—63% to their 51%, with ($p = 0.00895$).

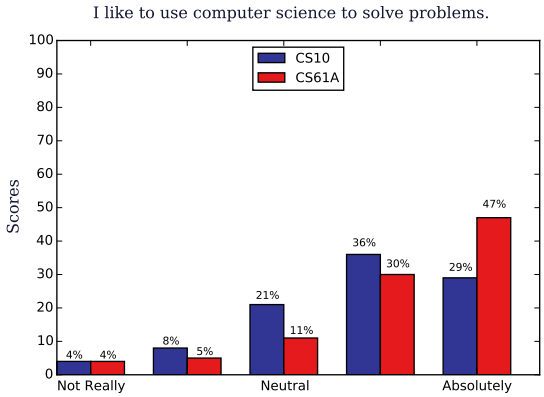
If we take it further and look at the belief of the female students, from the data in figure 6.16d, we see even more of a difference in favor of CS10, with ($p = 0.00030$). For female students who are without prior exposure to CS, in CS10, 59% of them believe they can get good grades in CS as opposed to 40% of the same cohort of students in CS61A. Even for female students who have had prior exposure to CS, still the cohort in CS10 have a stronger belief of achievement in CS in comparison to their peers in CS61A—75% with a strong belief versus 68% in CS61A.

[1] This figure was calculated by combining the percentages of respondents who exist in the spectrum between "Neutral" and "Absolutely."

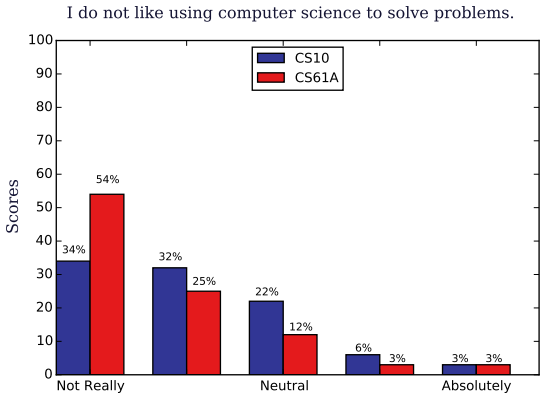
On Belief Around Learning CS Concepts

When we turn our analytical lens towards assessing students' self-reported beliefs around their ability to *learn* CS concepts, as usual we find stronger beliefs in students with prior exposure to CS, as we can glean from the data in figure 6.17, with ($p = 0.00064$). Almost three-fourths of all students in this cohort belief they can learn CS concepts.

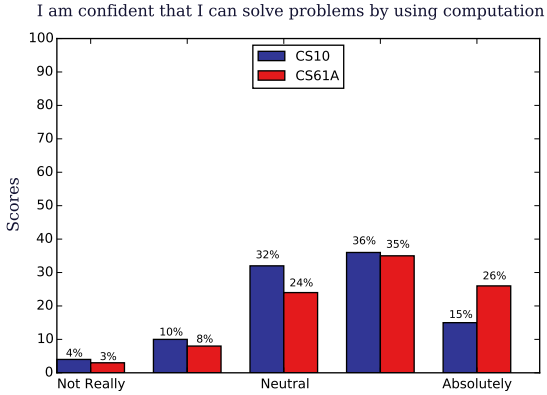
For students without prior exposure to CS, we find a slightly different story, particularly for the male students. From figure 6.17e, the data suggests that if you are a male student without prior exposure to CS, you are better off taking CS10. 71% of these male students have a higher self-reported belief about their capacity to learn CS concepts, than their peers in CS61A, who come in at 53%. Furthermore, 40% of the male students without prior exposure to CS, who participated in this study were neutral in their belief about their capacity to learn CS concepts.



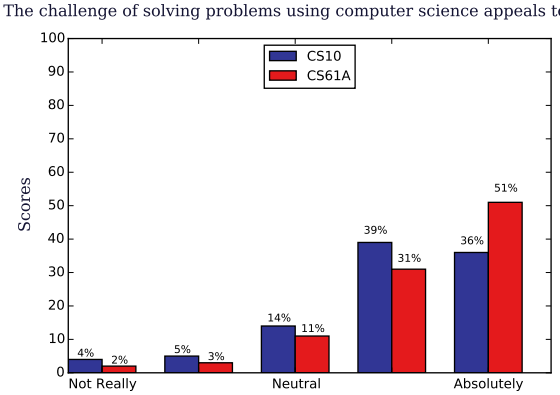
(a) atcs_1



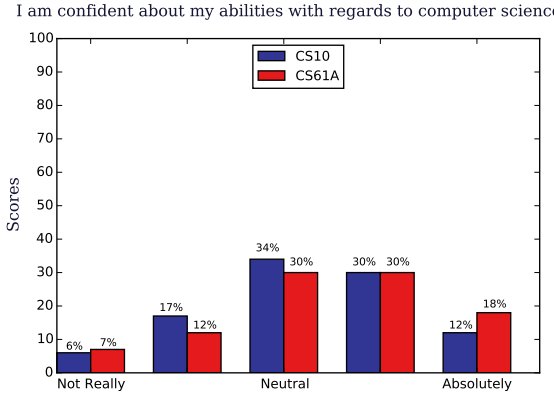
(b) atcs_4



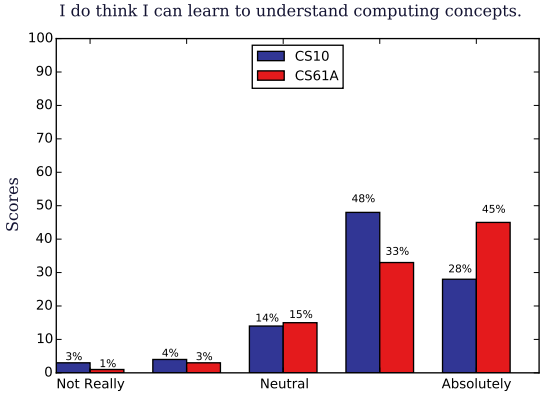
(c) atcs_5



(d) atcs_6

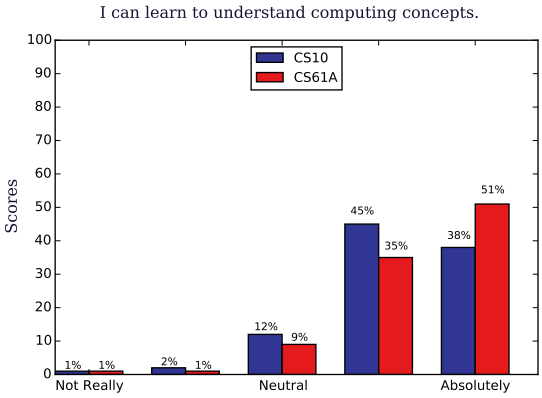


(e) atcs_8

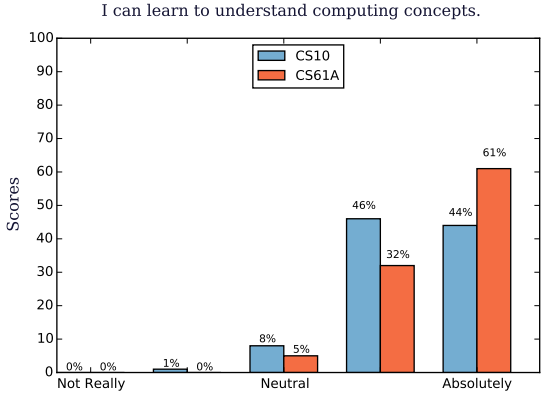


(f) atcs_9

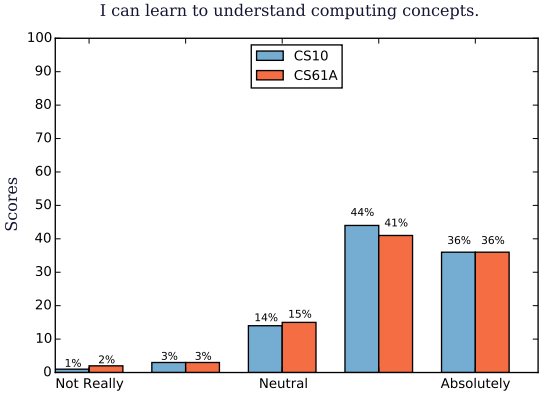
Figure 6.14: Students' Responses with Regards to CS Efficacy.



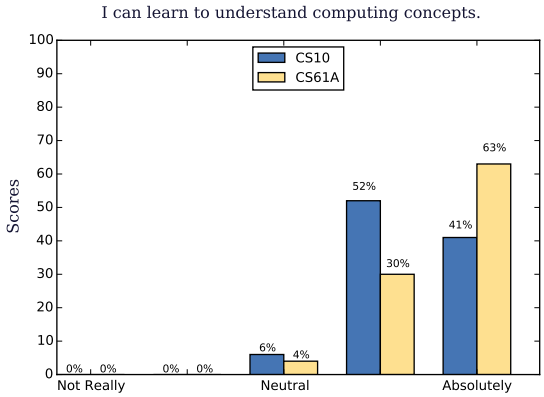
(a) atcs_2



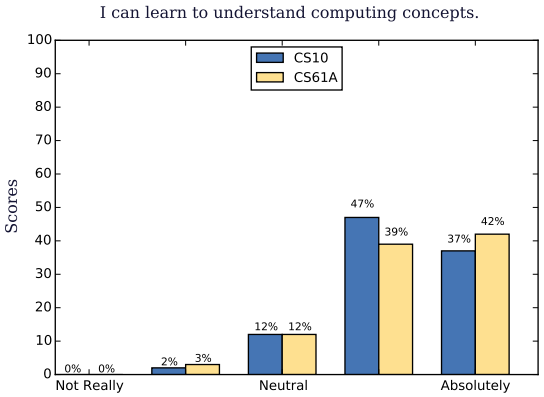
(b) Respondents with **Prior CS**: atcs_2



(c) Respondents with **No Prior CS**: atcs_2

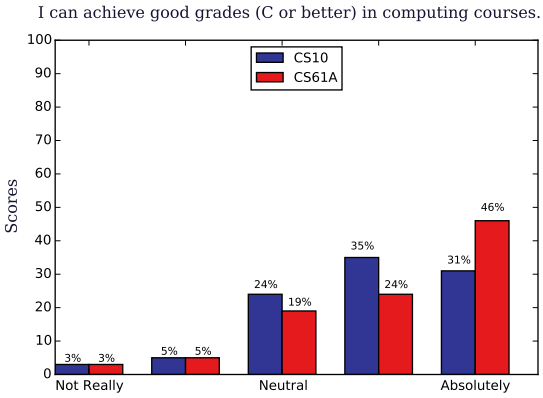


(d) **Male** Respondents with **Prior CS**: atcs_2

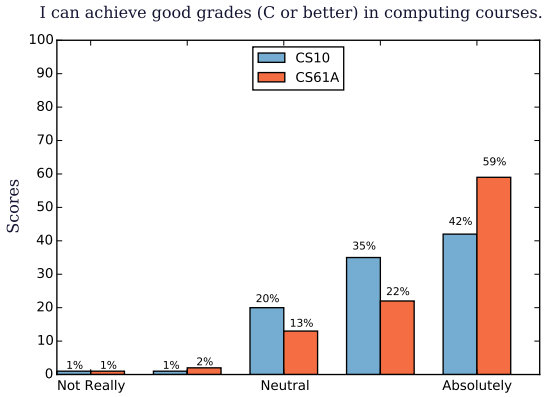


(e) **Male** Respondents with **No Prior CS**: atcs_2

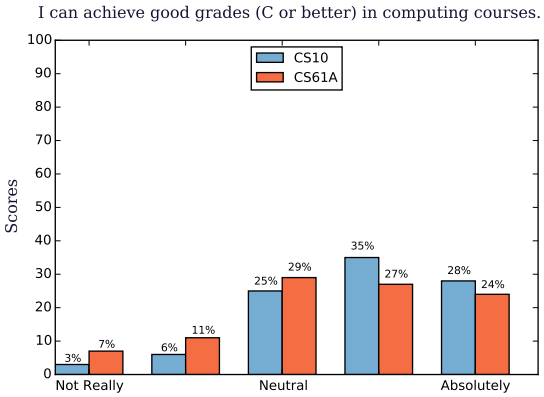
Figure 6.15: Students' Responses with Regards to Understanding Computing Concepts.



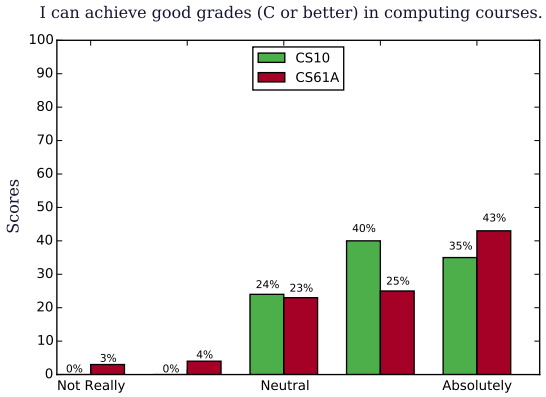
(a) atcs_3



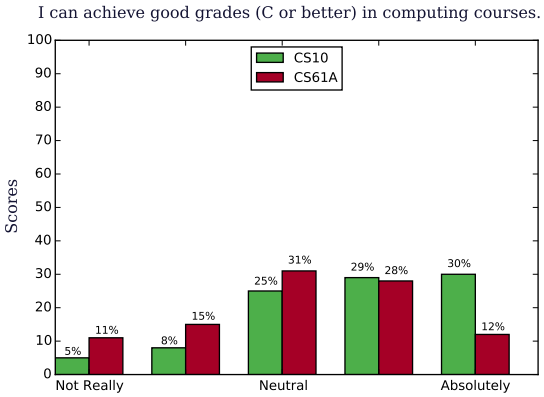
(b) Respondents with **Prior CS**: atcs_3



(c) Respondents with **No Prior CS**: atcs_3

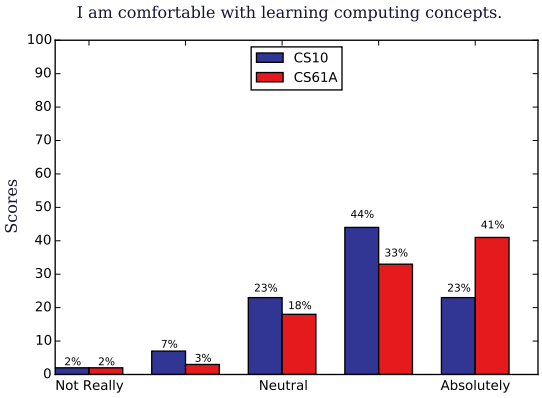


(d) **Female** Respondents with **Prior CS**: atcs_3

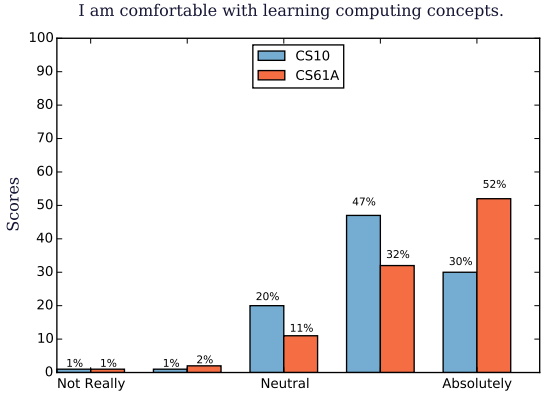


(e) **Female** Respondents with **No Prior CS**: atcs_3

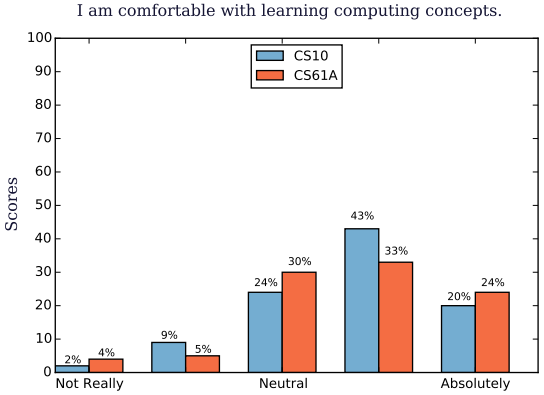
Figure 6.16: Students' Responses with Regards to belief around CS Achievement.



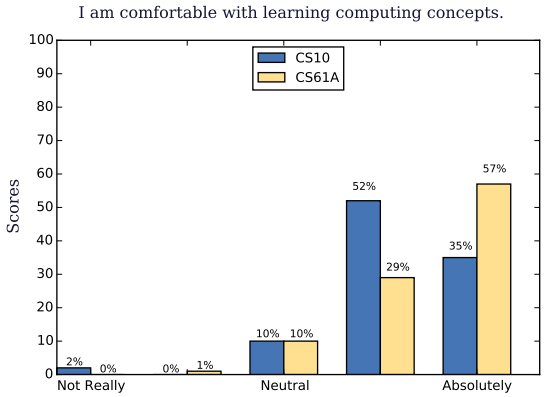
(a) atcs_7



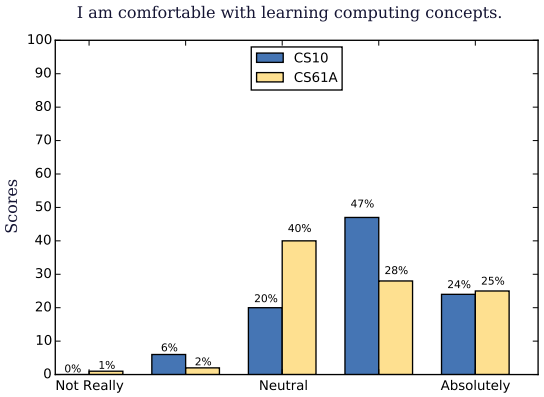
(b) Respondents with **Prior CS**: atcs_7



(c) Respondents with **No Prior CS**: atcs_7



(d) **Male** Respondents with **Prior CS**: atcs_7



(e) **Male** Respondents with **No Prior CS**: atcs_7

Figure 6.17: Students' Responses with Regards to Belief about Learning CS Concepts.

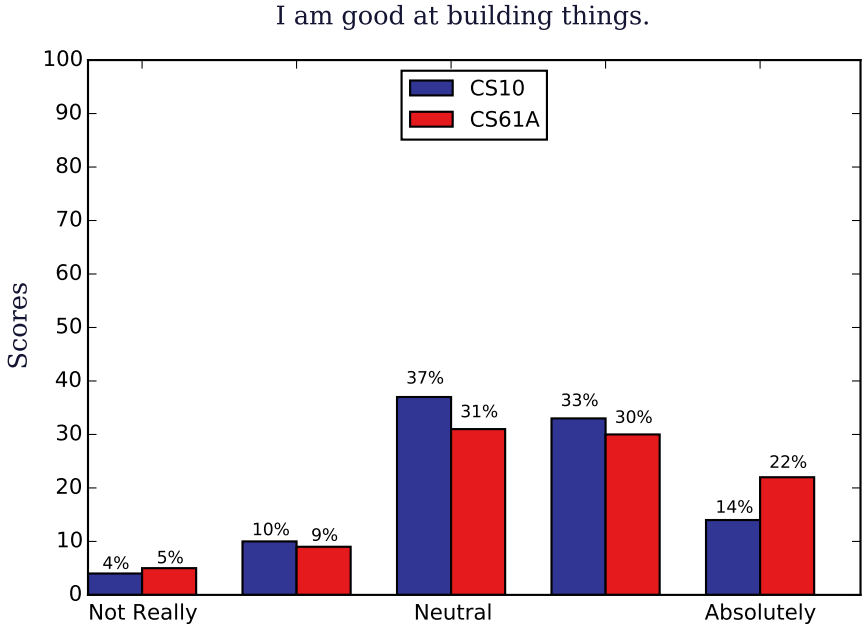
Findings:
On Programming

On Programming

Chief characteristic that I believe is indicative of an affinity towards computational thinking is an interest in building things, the skill of programming, and most importantly, the skill of programming as applied to problem solving. These three ideas were created and coded as follows:

atct_5 I know how to write computer programs.
atct_6 I am good at building things.
atct_8 I know how to write a computer program to solve a problem.

While I thought there would be some major differences on the experience of students in CS10 from that of students in CS61A when measured on atct_6, there were no statistically significant difference in their responses, as we can see from 6.18a.

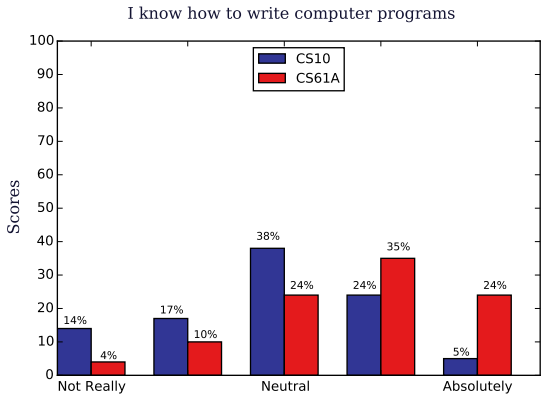


(a) atct_6

Figure 6.18: Students' Responses with Regards to tinkering.

What *was* interesting was findings around writing programs and writing programs to solve a problem. For instrument atct_5, about programming, 59% of CS61A students answered positively, versus 29% for CS10. This finding was statistically significant as can be seen from table 6.3, with ($p = 0.0000$). This means that there is absolutely zero percent chance that the data could have happened by random.

If one disaggregates the data around prior exposure to CS, looking at figures 6.20a and 6.20b, the effect is still there ($p = 0.00627$), but now, not as pronounced.



(a) atct_5

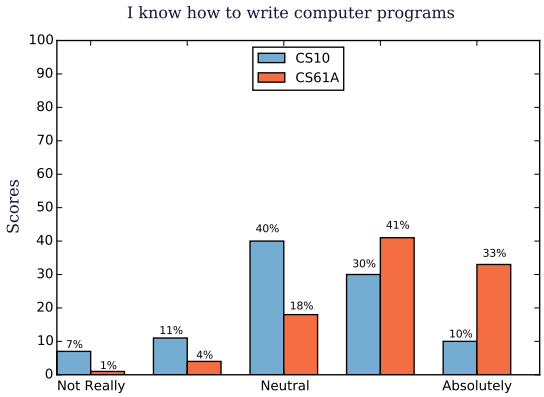
Figure 6.19: Students' Responses to Belief about Programming.

However if one further disaggregates the data based on gender, then one sees the effect vanish for female students without prior exposure to CS about their programming confidence in CS10 and CS61A. From table 6.3, we see that the statistical effect is present for both male and female students with prior exposure to CS.

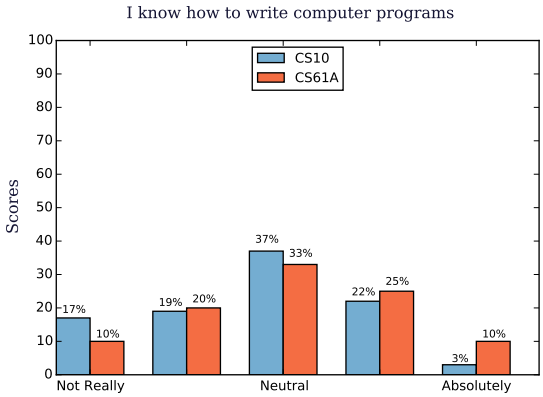
In Figure 6.21, I reduced the dimensions of the likert scale responses from five to three, by combining the two dimensions that were to the left and to the right of "neutral." This gave a clearer view of what the data looked like.

41% of the female students without prior programming experience in CS10 did not feel confident about their programming ability as one can observe from 6.21d. There were similar responses in CS61A.

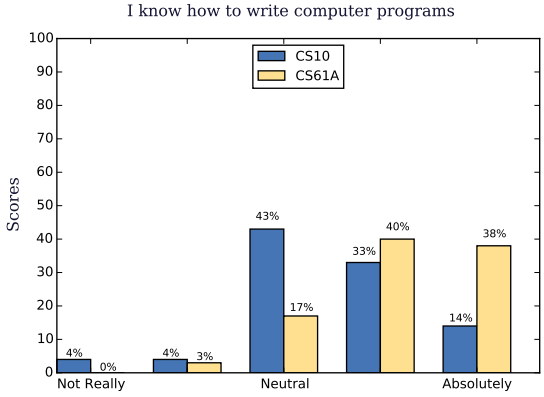
When asked if they could write a program to solve a problem, similar answer patterns were found. There was a clear effect ($p = 0.000$) in favor of CS61A—figure 6.22. However, in that case, there were no statistically significant effect on the experiences of female students in both classes.



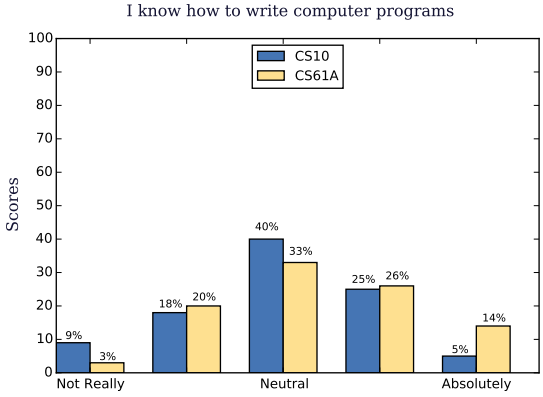
(a) Respondents with **Prior CS**: atct_5



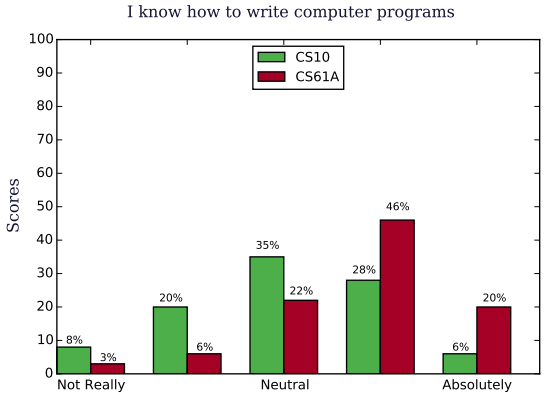
(b) Respondents with **No Prior CS**: atct_5



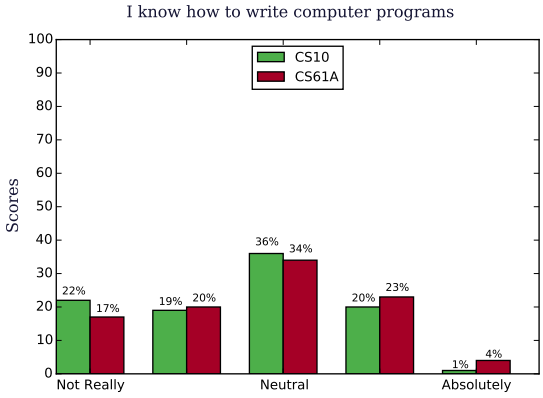
(c) **Male** Respondents with **Prior CS**: atct_5



(d) **Male** Respondents with **No Prior CS**: atct_5

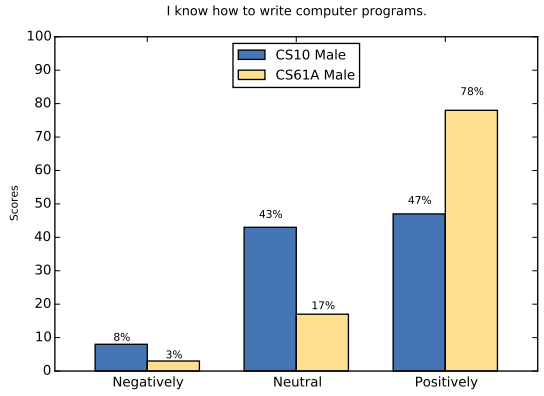


(e) **Female** Respondents with **Prior CS**: atct_5

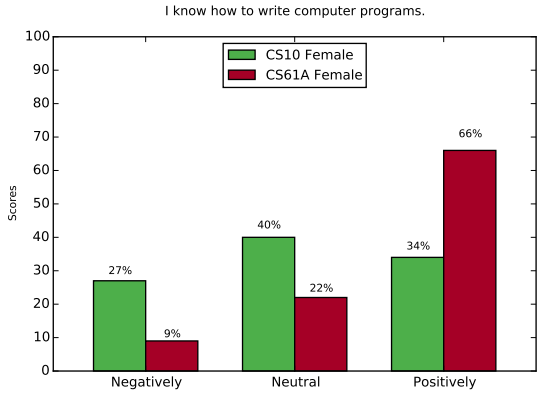


(f) **Female** Respondents with **No Prior CS**: atct_5

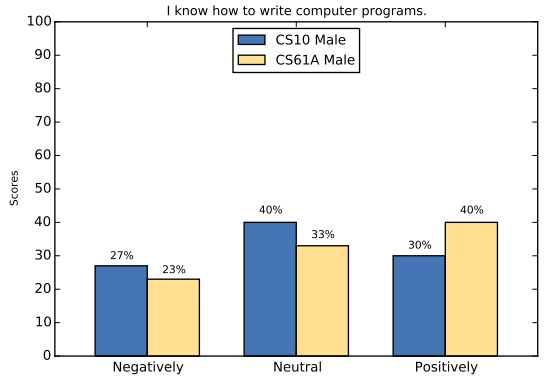
Figure 6.20: Students' Responses to Programming Confidence, (Data Disaggregated).



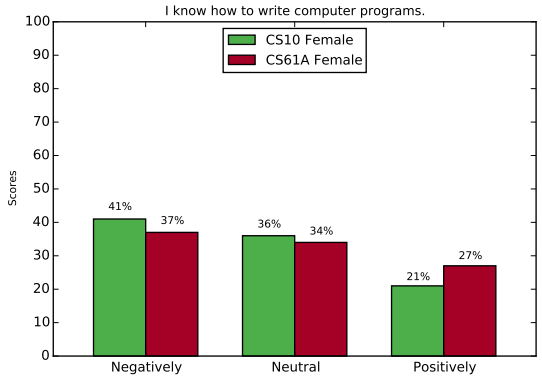
(a) Male respondents with **Prior CS**: atct_5



(b) Female respondents with **Prior CS**: atct_5

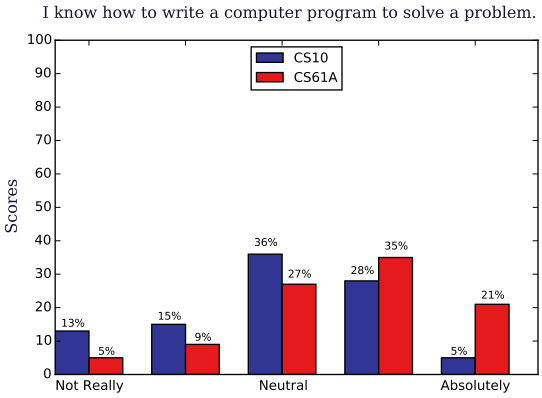


(c) Male respondents with **No Prior CS**: atct_5

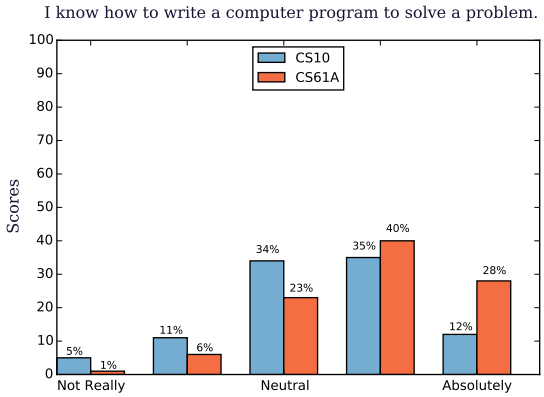


(d) Female respondents with **No Prior CS**: atct_5

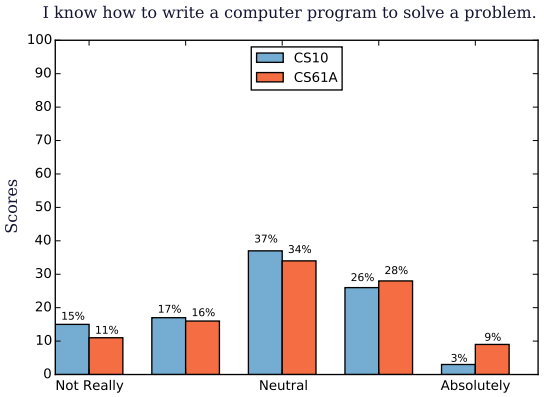
Figure 6.21: Students' Responses to Belief about Programming Confidence, (Data Disaggregated).



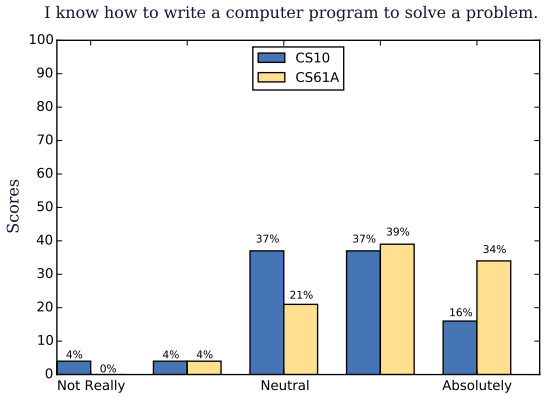
(a) atct_8



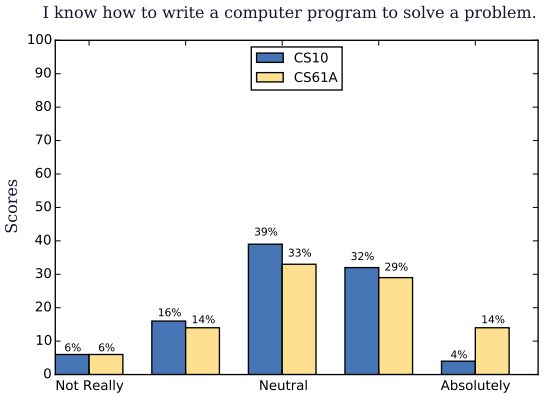
(b) Respondents with **Prior CS**: atct_8



(c) Respondents with **No Prior CS**: atct_8



(d) **Male** Respondents with **Prior CS**: atct_8

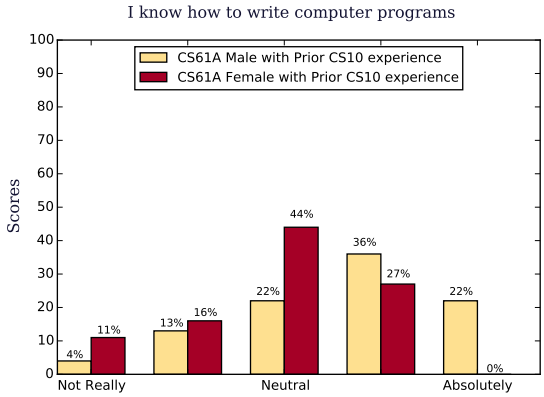


(e) **Male** Respondents with **No Prior CS**: atct_8

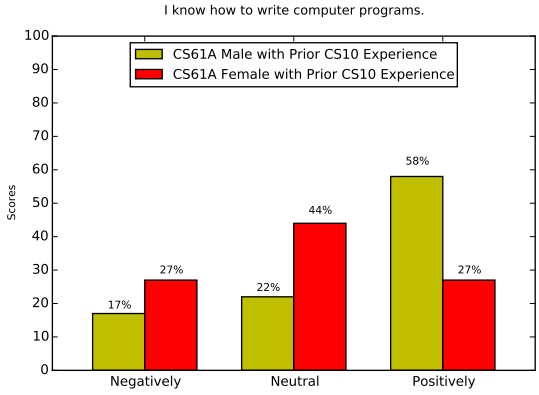
Figure 6.22: Students' Responses with Programming to Solve a Problem.

Effect of CS10 on CS61A Experience: Programming

To further get a clearer grained picture, I created a data subset of CS61A students who had previously taken CS10, to see if that intervention yielded a statistically significant effect on their experience. The effect for this instrument was significant with ($p = 0.04196$). When we reduce the dimension of the likert scale responses, we see that the confidence gap is *still* present as you can see from figure 6.23, and that only 27% of female students feel confident about their ability.



(a) **atct_5**



(b) **atct_5**

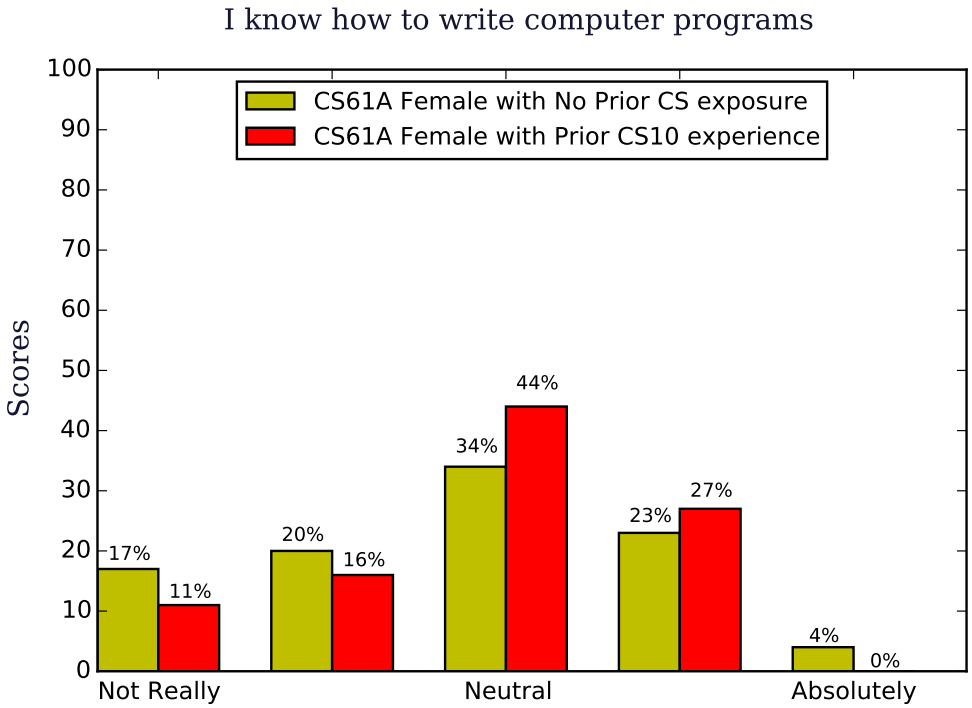
Figure 6.23: Effects of CS10 on CS61A Students' Self-Reported Programming Belief.

A side-by-side analysis was conducted as laid out in table 6.4 of female students who had previously taken CS10 and advanced on to CS61A, versus female students in CS61A with no prior CS exposure.

The result showed no statistically significant effect on the computational thinking dimension.

No prior cs exposure →CS61A
No prior cs exposure →CS10 →CS61A

Table 6.4: Experiment Set Up



(a) **atct_5**

Figure 6.24: Effects of CS10 on CS61A Students' Self-Reported Programming Belief.

Findings:
On Impact of Data Module

The Impact of Data Module

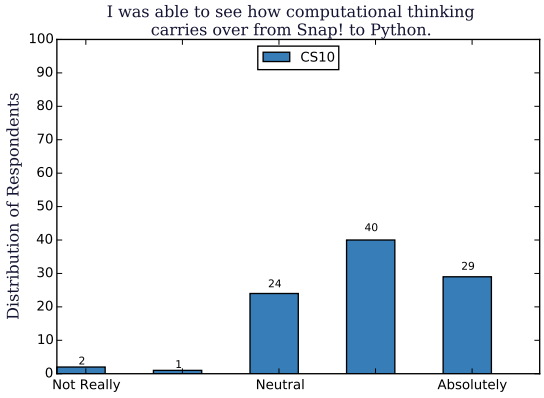
A smaller subset of CS10 students took this lab with N = 135, where N consisted of 62 male students and 73 female students. To determine if the lab had its intended effect, four survey items were created and coded as follows:

snap_python I was able to see how computational thinking carries over from Snap*!* to Python.

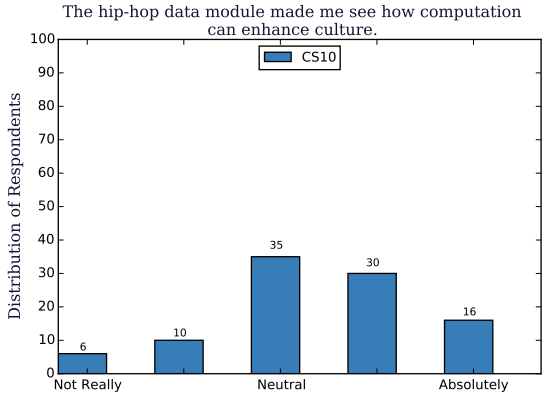
hiphop_d1 The hip-hop data module made me see how computation can enhance culture.

hiphop_d2 Lyrical analysis gave me a new way to think about computation.

song_ct I was able to build on my pre-existing knowledge about songs to further understand computation.



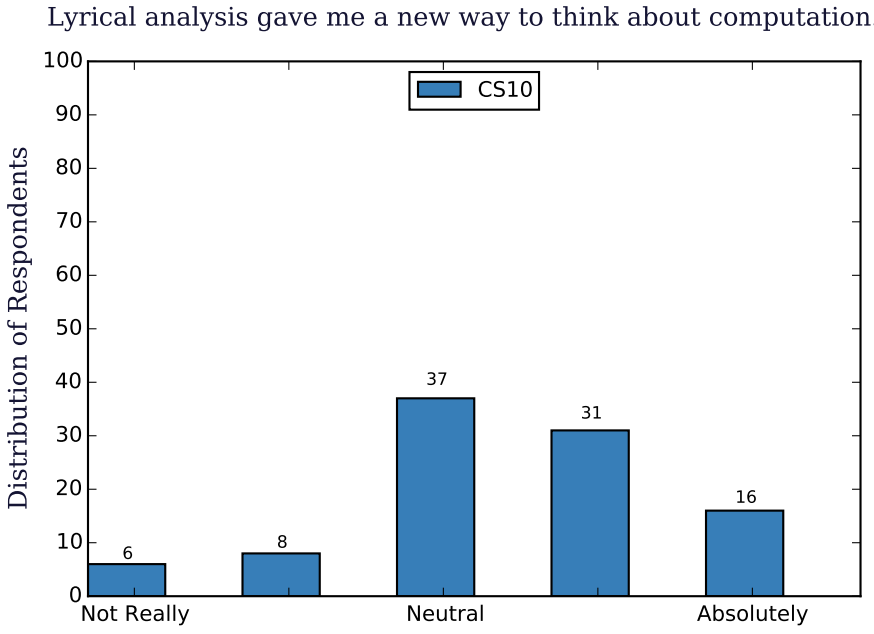
(a) Students' Response to Snap*!* to Python.



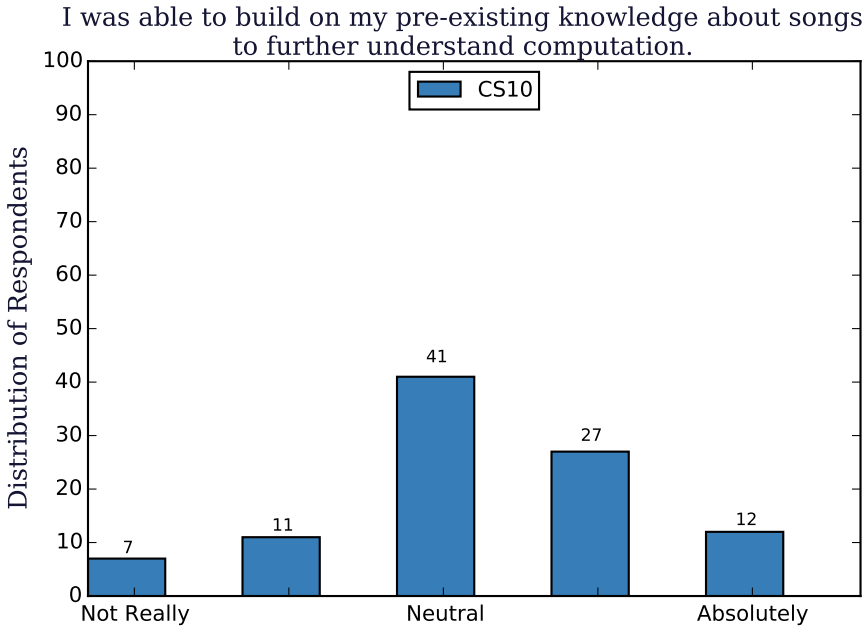
(b) Students' Response to Computation and Culture.

When asked about the labs ability to help them see how computational thinking carried over from Snap*!* to Python, 69% of students answered positively, as can be seen from figure 6.25a.

When asked about computation enhancing cultural knowledge, specifically by instrument coded "hiphop_d1," from 6.25b, we can see that only 46% of students reported that it did. Similar response were given for "hiphop_2," and "song_ct."



(a) Students' Lyric and Computation.



(b) Students' Song and Computation.

6.3 Qualitative Data Analysis

The quantitative analysis in the previous section has given the opportunity to have a landscape view on the experience of undergraduate students in introductory CS at UC Berkeley. Since this is a mix-method research approach, moving forward I broaden my investigation by conducting a qualitative analysis of a subset of the population to get at specific lived experiences of the participants within the context of the classes under study.

Of the 24 interview participants most respondents were undergraduate students, of which 13 were female and 14 were male. Care was taken to get a nice cross section of students who had experience in CS10, CS61A and both. A breakdown of participants can be found in table 3.5.

I transcribed the in-person audio-recorded interviews and read through to see what were some emerging themes. A text analysis was conducted on chunks of interview text that corresponded to each question asked from the interview protocol that can be found in appendix D. Technical implementation of the analysis can be found in appendix B.2. Each discrete question and its associated probes was assigned a code that can be found in table 6.5.

Primary Level Coding

The initial thematic coding phase was completed through *Frequency* coding, i.e., applying a key phrase extractor algorithm to interview transcripts using the Natural Language Toolkit (NLTK) package for the Python programming language. This yielded a series of salient phrases and words that represent memo-codes as can be see in table 6.6.

The following eight labels were assigned as a result of the memo-code: Programming, Family, Work Style, Minority Experience, Belonging, Efficacy, CS Efficacy, and Perception. These labels represent the various aspects of undergraduate CS experience that indicate a significant correlation to the decision to explore CS, and stay with that decision. You can observe the key words extracted from the raw data which correspond to each of the memo-codes in table 6.7.

Table 6.5: Coded Interview Questions

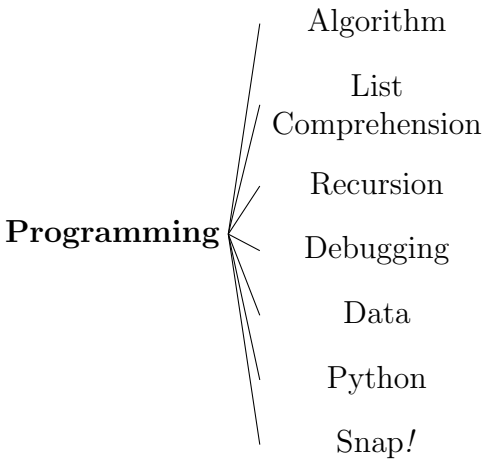
Code	Interview Question
currFactor	Were there any curricular factors in the course that stood out to you?
enjoyCourse	Did you enjoy the course?
extra	comments and remarks.
feelMinorityCS	Do you ever feel like a minority in any way in the CS department?
funAspectClass	What aspects of your experience did you find particularly fun or rewarding?
howLearnComp	Tell me how you learned about computers?
magicWand	If you could sort of have a magic wand, and I told you, I want you to help me figure out how to get more women and minorities in computer science at Cal[2]. And I told you that this was the problem I would like you to help me solve, What would you do?
negAspectComp	In your opinion, what are positive and negative aspects of computers?
probStudyCS	What do you perceive as problems in studying CS?
respectCourseStaff	Did you feel respected by the instructor?
strengthCS	What strengths do you think a successful computer science major should have?
unfunAspectClass	What aspects of your experience did you find unfavorable or frustrating?
courseCulture61A	How would you define course culture of CS 61A?

Table 6.6: Memo-Codes

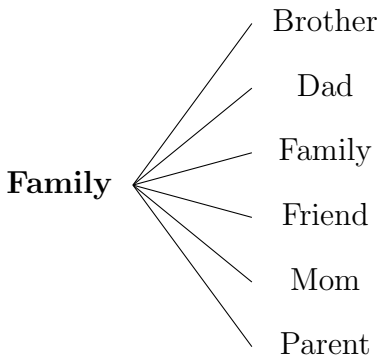
Interview Questions	Extracted Key Words	Memo Codes
currFactors	algorithm, list comprehension, project, recursion.	– Project, Programming
enjoyCourse	Autograder, game, a-lot, project, yes.	– Self-directed projects/-Efficacy
howLearnComp	Apps, APCS, brother, dad, engineer, engineering, family, friend, freshman, mom, parent, program, school.	– Family, Efficacy, Profession
strengthCS	Ability, collaboration, creative, debugging, good, help, learning, logic, math, people, problem, project, skill, strength, time.	– CS Efficacy, Work Style
feelMinorityCS	Asian, Berkeley, class, community, different, everybody, everyone, experience, feel, girl, guy, a lot, lab, minority, partner, people, project, white, woman, yes.	– Racial experience, Gendered experience, Belonging
funAspectClass	Data, Denero, different, job, lab, people, programming, project, real, science.	– Data, Cultural Relevance, Programming, Belonging
unfunAspectClass	Capability, code, experience, grade, hour, point, project.	– Efficacy, Extrinsic Motivation, CS Efficacy
courseCulture61A	Community, course, everyone, help, lot, many, office-hours, partner, party, people, project, question, TA.	– Belonging
respectCourseStaff	Lab, peer, people, person, problem, professor, student, TA, way, yeah, yes.	– Belonging
extra	Codecademy, culture, hackathon, lab-assistant, programming, project, python, recursion, snap, transition.	– Transition from novice to advanced beginner, Online supplements, Culture
magicWand	Image, people, perception, problem, men, stereotype, student, woman.	– Perception, Gender

Table 6.7: Words Indicating Memo-Codes

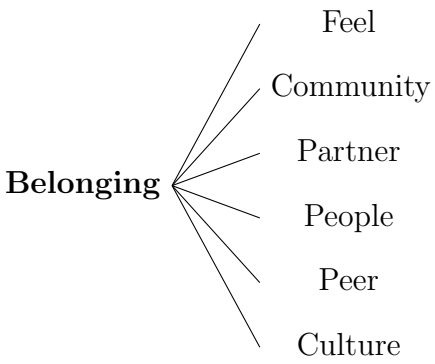
(a) Words Indicating Programming Concept



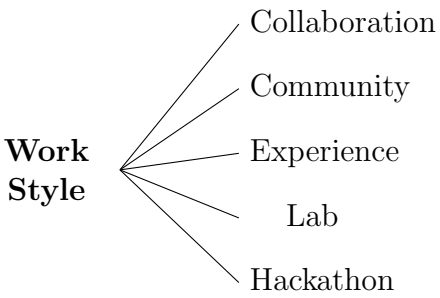
(b) Words Indicating Family Concept



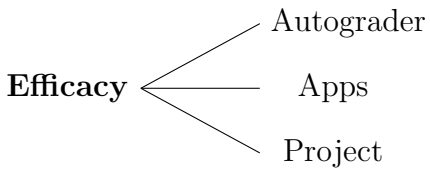
(c) Words Indicating Belonging Concept



(d) Words Indicating Work Style Concept



(e) Words Indicating Efficacy Concept



(f) Words Indicating CS Efficacy Concept

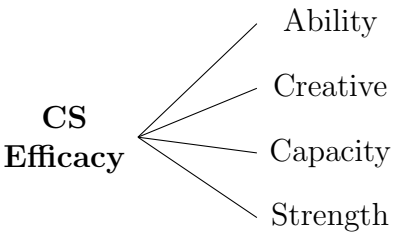
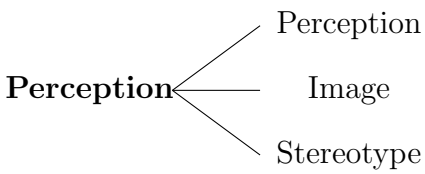
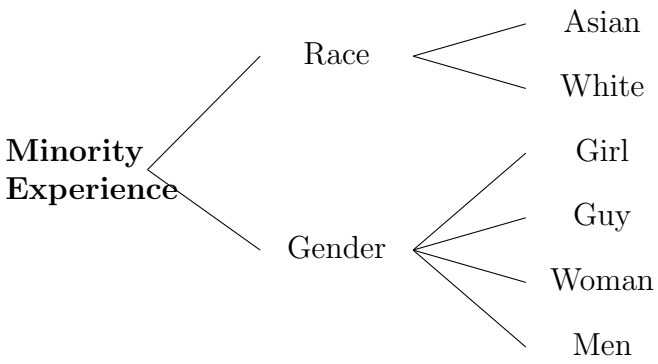


Table 6.8: Words Indicating Memo-Codes (*continued*)

(a) Words Indicating Perception Concept



(b) Words Indicating Minority Experience Concept



Secondary Level Coding

The second-level coding phase of the data analysis process, *Pattern* coding, resulted in a meta-code of six categories ascribed to the eight initial labels derived from the interview transcripts as shown in table 6.9.

Table 6.9: Meta-Codes

	Code	Defintion
1.	**Outsider Experience**:	Initial level coding labels Perception and Minority Experience were combined to derive this meta-code.
2.	**Programming**:	Initial level coding label Programming was promoted.
3.	**Role of Family in CS Choice**:	Initial level coding label Family was promoted and renamed the meta-code"Role of Family in CS Choice".
4.	**Belonging**:	Initial level coding labels Belonging and Work Style were combined to derive this meta-code.
5.	**CS Efficacy**:	Initial level coding label CS Efficacy was promoted.
6.	**Efficacy**:	Initial level coding label Efficacy was promoted.

Outsider Experience

The data analysis revealed that many of the ethnic minority students in the study got into CS because of a previous exposure to the field; specifically through various out of school programs targeted at ethnic minority students. One participant of color stated that

> "I did this program during high school called smarts at UCLA where they like teach people like you know about the different engineering fields. I had see like what java was but I didn't really know how to do it, so I was like okay, I guess I am going to do computer science."

Padme noted that the UCLA program had a significant impact on her decision to major in computer science.

With regards to their experience of being historically underrepresented in the major. Many participants were acutely aware of their predicament. Some had a stronger sense of being a racial minority than being a gender minority. One participant said:

> "Not so much in CS10, and the rest of the 61 maybe, not so much for being a girl but more for being *White*, to be honest. Its a very heavy Asian majority. There's a lot of international students, also very heavy Asian or Indian majority. Include in the TA's. I find it very interesting, if you look at the staff, between CS 10 and the 61 series, there is a very big dichotomy. With CS 10 staff, it is very interesting because it's a lot of Caucasians and half girls also. You go to the rest of them, they are mostly Asian."

Another participant spoke about how the negative stereotype around CS almost had him turned against *even* having the vague thought of trying it out. During his interview, he stated:

> "It just looks so scary and hard, but CS10 was really approachable, pretty blocks. People don't really know what programming languages are. It is very intimidating, especially what you see on media, where you see all these whiz kids just typing away, the crazies, it's your screen full of 1s and 0s. I think a lot of people think you actually code in binary."

That participant eventually tried CS10 after his friends convinced him that it wasn't what he thought it would be.

From the analysis, the realization that some participants had about their minority status wasn't enough to deter them from exploring the field, many developed coping mechanisms to deal with their status.

Role of Family in CS Choice

The data analysis revealed that first degree family members play a significant role in introducing the idea of CS to participants. From the analysis I found that most students took CS10 because a friend had taken it, or a family member had introduced the idea of CS to them. For CS61A the findings were similar, except there were more students who had taken APCS in high school, which led them to the class once they got to the University.

To drive home how important these first degree family relationships and friendships are in the decision of a student to try CS, one particular participant shared how his his friends led him to take CS, even though he had decided before he *even* got to Berkeley to *not* participate in anything related to the major. He states:

> "I came in a Math major, refusing to learn to program, my roommate had taken CS10 the first semester, and by then he was a reader, I took it and it was a fun, he was staff and I became a staff too."

A female graduate participant shared how she used to do projects with her father, who introduced her to the field. She states:

> "I did really small projects with my Dad who is a computer scientist, and I had had a CS course in undergrad but I didn't learn very much from it. "

In her case, her undergraduate experience was difficult, so she departed from the major, only to re-examine it again when she got to graduate school at Berkeley.

Efficacy

In analyzing the data associated with undergraduate student's experience of introductory CS at Berkeley, their ability to engage in self-directed, culturally relevant projects in CS10 was a consistent aspect that was raised.

Specifically, lack of open ended projects seem to reduce enjoyability of the experience for CS10 students who advance on to CS61A. One participant shared the following when asked "Did you enjoy the course?":

> "CS10. Yes! I did enjoy 61A. I don't actually know if I enjoyed it as much as CS10, but yes I did enjoy some of the concepts in 61A . . . one of the things people really enjoyed in CS10 is that the projects are really open ended and you can make whatever you want. In 61A, I felt like a lot of the project for me was like bashing against the autograder, its like do this, and then do this, and then do this and there is an autograder check all the way. So the projects are kind of like I don't even necessarily know what I am doing, you are just kind of like pushing through the autograder. CS10 is nice because, you come up with the idea for your project, and then you completely design the plan for this project yourself and you are able to submit that. That is one of the things I enjoyed more in terms of CS10."

This insight help develop the working hypothesis that one of the reason why CS10 students don't do as well in 61A is because of a lack of open-ended projects. Through the data analysis, many participants also brought up their displeasure at the grade focused orientation that exist in CS61A, which they said was not present in CS10.

Belonging

The analysis revealed that undergraduate students in the introductory CS pipeline at Berkeley understand the role that fitting in, or being accepted by their peers has on their learning experience.

Many students especially in CS61A have to rely on their peers to make it through the class. One participant spoke about how important having community is, in determining whether one will find its easy to excel in the major. She shared her take on the climate of major in this following excerpt from her interview:

> "The thing that I find that it's maybe better in Berkeley, because we have such a large computer science major, there is a lot of support, but at the same time it is kind of difficult especially with big classes like 61A, no one is going to hold your hand, you have to find friends. Even going to office hours sometimes does not work because there's too many people. So you really have to fight through it by connecting with your peers . . . It could be a good thing if you like that kind of thing, it could be bad thing if you're not comfortable doing that kind of thing."

6.4 Triangulation

Corroboration was proven in the data analysis through the process of triangulation. The extracted keywords from the initial raw data were re-examined to make sure they constituted the right meta-codes.

There was not a need to code and recode the initial raw data as is done in orthodox qualitative data analysis because the automated keyword extractor software eliminated subjective errors.

Meta-codes developed from the secondary coding level in combination with the various dimensions developed through quantitative data analysis show a significant correlation to the recruitment and retention of undergraduate students in the introductory computer science pipeline at UC Berkeley.

In conclusion, this chapter has presented the findings from the mixed-method study of the introductory to computer science pipeline at Berkeley. In understanding why students choose CS the following dimensions have been identified: Mentorship, Family, Outsider Experience, Belonging, Programming, Efficacy, CS Efficacy, Social Implications of Computing, Gendered Ideas of Intelligence and Transitional Data Module.

Chapter 7

Discussion

In this chapter, I will expatiate on the findings from this research and discuss the implications of those findings in answering the four research questions at the heart of this study. In addition, I give my final conclusion and recommendations on how we can move towards equalizing participation in CS in a sustainable way.

7.1 Discussion

Research Question R1.)

What are the social factors that contribute to the underrepresentation of historically marginalized groups in CS?

In answering this question, this research discovered some rather interesting things. First, when testing was done to measure students self-reported beliefs about their CS efficacy, the research found that CS61A students had a higher self-reported score on almost all dimensions, as compared to CS10 students, except for the instrument coded "atcs_3: I can achieve good grades (C or better) in computing courses."

Focusing on the experience of students without prior CS exposure, we found that CS10 students had a stronger statistically significant belief in their ability to get good grades in CS, than their peers in CS61A, who had not have a prior exposure to CS. This findings is particularly apt for female students.

Along similar lines, when we asked students about their ability to *learn* CS, we found that male students who are taking their first CS course on Berkeley's campus, those student who take CS10 instead of CS61A, have a stronger positive belief around their capacity to learn CS concepts. This finding corroborates our belief that CS10 is a better class for the non-affiliated CS person to take as their first taste of the major.

Second, while students clearly *reject* that women are less capable of success in CS than men,

the jury is still out when it comes to the myth of innate intelligence based on gender. As much as we would like to dislodge gendered notions of intelligence, the data displayed in 6.5b show that these ideas are still inconclusive in the minds of our young people. Particularly female students in both CS10 and CS61A are *still* undecided about whether "women are smarter than men," as can be seen in the data plot of figure 6.7c and 6.7d.

With that said, it comes as no surprised when testing was done to measure students self-reported beliefs about their CS efficacy that male students had a higher score that female students on almost all measures. From Buse, *et al.*, we know that women who persisted on the engineering track in industry had high levels of self efficacy and optimism. Where as those that left showed high levels of self-doubt in relationship to the engineering track (Buse et al., 2013). From Cheryan, *et al.*, we know that:

> "Environments can act like gatekeepers by preventing people who do not feel they fit into those environments from ever considering membership in the associated groups."(Cheryan et al., 2009)

This and other scholarly research help form a working hypothesis that learners are more apt to identify with a field when they feel a sense of belonging. This fact is ever so illustrated by a conversation I had with a participant I will call Jane. Towards the end of our interview session, I asked her if there was anything she would like to share before the interview was over, she replied with the following response:

> "Jane: I recently talked to someone who is now a computer science major and a junior actually but she told me this really interesting story that was like eye-opening.
>
> I asked her what from CS10 made her want to keep going? And she said she came into Berkeley like an MCB major and she just took CS10 for fun, her room mate was in the class and stuff, and for the final project they got make whatever you wanted I guess.
>
> And so, she said she had this idea and she went to her TA, told him I want to make this thing that did blah blah blah, and **the TA to her like that is awesome go build it!** And she said no one had ever told her that before.
>
> Like no one had ever propelled her to do these things, or make things, or use her creativity to create things. She was like, she's never gotten that in school before. Because of that, she wanted to keep going. Then I was like it must have been an eye opener when you went to 61A and they tell you everything you have to do and you never get to be creative like until your upper divs 2 years later."

From the excerpt that Jane shared, we can see how encouragement from an authority figure can lead female students to a stronger sense of belonging. And how that in turn, can make them want to stay on in the major.

From the research findings, when assessing students sense of belonging, focusing on ideas, *still* only about 50% of female learners felt their ideas were important in the class ($p = 0.00028$). This means that nearly 50% of female learners felt like their voice wasn't important. You can see this in figure 6.11f. To be fair, similar proportion of boys also felt like their voice didn't count as well.

The observation of the 30%+ of male students that feel a lower sense of belonging, goes to show that CS isn't particularly a "male" field, certain male learners *still* feel like they don't belong in it. At best, our introductory to CS classes at Berkeley are only igniting about 50% of male learners. The data supports the findings of Cheryan, *et al.*, who remind us that:

> "Because the importance of belonging is ubiquitous (Baumeister & Leary, 1995; Fiske, 2004), we suggest that even those who are members of well-represented social groups are susceptible to feeling a lack of belonging in an environment that is not compatible with how they see themselves." (Cheryan et al., 2009)

This makes our emphasis on equalizing participation in CS, even more crucial. While heretofore, a lot of work has been done in understanding why historically underrepresented minorities disengage with CS, there hasn't been significant work done in understanding what makes "anyone" see themselves as a CS kind of person.

The introductory CS pipeline at Berkeley, as defined by CS10 and CS61A, still makes around a third of students feel distant in terms of their sense of belonging in the classes. This sense of alienation is statistically reduced in CS10 versus CS61A. However, previous participation in CS10 before attending CS61A, doesn't insulate female students from a negative sense of belonging in CS61A.

It is my belief that we see this reduced sense of alienation in CS10 students because, one, the class size is significantly smaller, 300+ versus 1200+, and two the class does a great job of connecting CS to the wider world. CS10 students are often asked to investigate and think about the impact software has had on the world, both the positive as well as the negative. Further, CS10 supports its students to think critically about their role in the world as technology creators. I believe this is what is responsible for the increase in the students sense of belonging in that class.

Another rather interesting finding from the research was along the lines of mentoring. Interestingly, male participants in the study felt that there were *not* a lot of people in CS industry that looked like them when compared to female students! You can see this from figure 6.2d. However, they had slightly more role models than the girls.

Given the fact that Silicon Valley tech is mostly White and Asian males, and Berkeley is also mostly White and Asian, this finding seems rather puzzling. Nonetheless, previous research does support the notion that students often have a gap in stereotype of the field and the reality of it as can be seen in (Fisher et al., 1997).

From the quantitative data analysis, we found that unsurprisingly, 90% of CS10 students had not taken a CS course in high school. From that we can see that the class is truly fulfilling its mission as a safe place where non-majors can explore the field. With that said, in addition from the data, we observed that at least 40% of female CS10 students had a first degree family member that was affiliated with the field.

From the qualitative data analysis, we found that many of the students tried the class either because of a friend, or because a parent had urged them to try this. This finding is supported by scholars who have noted the role of parental support in choosing CS. They state,

> "We found that having parental support, especially maternal support, was a greater influence for women in computing than their male counterparts."(Redmond et al., 2013)

One participant particularly shared how her mother had urged her into the major, because her mother had missed her opportunity to major in CS. Willow states,

> " Interviewer: Do you have any parents at all that are related to the computer science industry?
>
> Willow: No, my dad is a mechanical engineer and my mother was an accountant. But my mom actually really liked computer science, but she never really learned a lot about it, but she used to work with some other tech guys at the company and she would look true lines of code with them because she was really interested in it but never got into it. So I think it was like her, when she told me to do it, 'like Willow, I never got to, so you should try it.' "

On the other hand, while Willow's mom was encouraging her to try CS, another participant was actively dissuaded by her mother to stay away from the field. When I asked Serena how she got interested in CS, this was her response,

> " Serena: Both my parents are in engineering, so I have that proximity to it. And I grew up around the Silicon Valley area, but not in it. My parents didn't want me to get into computer science, what really drew me was I never really had a laptop until 10th grade, I just found an old laptop and fixed it myself, I just got really really drawn to computers learning how they work.
>
> Applying to colleges, I just applied to schools that had computer science, one because my parents wanted me to be in a field where I could make money. I kind of wanted to to go into something in that arts but, they wanted me to be in the medical field, so I guess I decided computer science could be happy in between.
>
> I could be excited about doing it as well as make money. I guess I really fell in love with it once I came to Berkeley, so here we are.

> Interviewer: So you said your parents didn't really want you to take computer science, can you elaborate on that?
>
> Because of my gender, my sex, whatever.
>
> Interviewer: Because of your sexual orientation?
>
> No because I'm female.
>
> Interviewer: Because you're female, even though both of them are engineers?
>
> Yes. yes.
>
> Interviewer: She, your mother did not want you to go down that career path?
>
> Yes. She didn't like it, it was really really stressful for her, also she expects me to one day raise a family, and sees the two as being really incompatible. In the tech field and sometimes raising a family can is sometimes incompatible, I don't know.
>
> She saw them as being incompatible. She does not work anymore, she got laid off during the recession, never got a job after that. I guess part of it was that she really, really, really, did not like it. And she did not want me to be part of that experience."

We see here how parental influence can both be supportive as well as deleterious to the desire for a female student to pursue CS. Willow's experience, was radically different from Serenas. Willow comes from a upper-middle class, White family. While Serena parents migrated from Vietnam. In her own words, she says about her background,

> "My parents were originally from Vietnam, they immigrated here, both of them, they went to college, they transferred from community college, they worked their asses off to get to where they are.
>
> I was born in Salinas, which is in central California, predominately Hispanic and middle-class and after that I lived there for three years and moved to Gilroy, same-ish demographic really really diverse friend group."

This excerpt shows that gender, class, as well as ethnicity plays such a major role in determining whether one is going to go forward in CS or not. These are factors that are well outside of our control as learning scientists. In our own little academic fiefdoms, we can manipulate a few variables here and there, to create a level playing field, but ultimately we are fighting against some enormous societal forces that need a collective response to address.

Willow, and Serena represent students on different radial axis that measures belonging based on a proximity to a CS norm, i.e., White and Asian, middle-class to upper-class, male. I want to introduce you to Rapunzel, a students who represents the "complete outsider." A student whose institutional conversion to CS will represent to us, that we have solved the problem

of equalizing participation in CS. I asked Rapunzel a pointed question about whether she felt like a minority, her reply to me was not what I at all expected.

> "[laughs] Am not even sure of how to explain it but I will try to make it short, but, I felt like a minority in that, how do I say it, I have examples but I am not, it would be kinda of long, so I am trying to ...
>
> I remember once in discussion, very early in the course, um, the TA brought in a computer for us to like take a look on the insides and the inner workings of it, they were people in the room that knew exactly what these pieces and parts did, the names and everything, and I had never in my life seen anything like that.
>
> I was just like, that was the first time I wondered should I even stay in this course [CS10] because I felt like they told us it was for people that had never had any experience in it. But we had people in there that obviously knew a lot more than I did, so I was also very unsure of that. There were people in there that had experiences with equipment that **I know I would never even come close to being able to afford.**
>
> Even dealing with macs, macs were very difficult for me to use because I had never used a mac before. Then we had gone to using only macs mostly, that even bringing my laptop, we eventually came to a point were they preferred we use the macs.
>
> I found that difficult, it was just, [laughs nervously], I don't know, it was so many different things, even being like, I know there were other racial minorities in the course, but there were none in my discussion or lab, so it was just complicated.
>
> There were quite a few things. "

Lets unpack what she just shared, one, she felt like she was out of her league. Several things are going on in that environment that makes Rapunzel feel this way. Two, she was probably the only ethnic-minority student in the room at the time. Beyond her race though, there are even more salient signifiers that she was without, which I will posit made her feel more uncomfortable than being a racial minority.

We often place so much emphasis on race, but when we talk about race in this context, it is quite *meaningless!* How can a human being come from a "concept?" The answer is, they can't. To label Rapunzel's experience as the quintessential African American experience is flawed. Instead, her experience should be labeled as the "outsider" experience. Lets switch gears for a minute and learn about Boba. I asked Boba to tell me a story of how he became acquainted with computers. Here is what he said,

> "The first memory of computers, I remember my Mom was always using one, she is pretty tech savvy, she is a software engineer.

> So when I was pretty young probably elementary school, we had computer class in elementary school, in terms of computers, those were really like my first interactions, nothing special I think. I wasn't in programming or anything, I was just using computers.
>
> I first started programming like in middle school I think, I went to a camp about computer programming in Michigan, I used to live there, I learned Java for a couple of weeks. I made like pong. They gave us a framework and I just did a little bit of customization on it and learned object oriented programming.
>
> I typed a little bit and tested out stuff until like 8th grade. I got an iPod touch and I played around with the apps on there and I was like see I really want to make one, and then learned I needed a mac, bought a mac and then just got a book and just learned to make iOS apps.
>
> I have been doing that like ever since basically. By 10th grade I was making my own apps. I guess its different from CS but its programming. "

For Rapunzel, her race isn't what makes her an outsider to this field. Its more her experiences, the rituals that indicate where she is from in terms of the human story. While for Boba who is Asian, his race isn't what makes him an insider to the field either, it is his experiences that overlap with the experience of a "complete insider."

These rituals contain so many signifiers that make one and insider or an outsider. A parent that works at Apple, the habit of caffeine, drinking certain beers, pulling all nighters, being part of the twitterati, having friends who have "exited," using terms like "exit," "raise a round," "dev" and so on. These are the rituals that separate the insiders from the outsiders.

For some students, the rituals of CS are familiar. They are the rituals they grew up with. Take Boba for example, he grew up fully ensconced in the rituals of tech, and in his case, privilege. While Rapunzel couldn't even phantom owning a mac, Boba decided he needed on in 8th grade, and like that he bought one.

To answer a question like what social factors contribute to the underrepresentation of historically marginalized groups? Is to understand how to systematically transition a person from an outsider to an insider. Access to solid curriculum plays a part, but the most important is acquiring the signifers of inclusion without damaging ones sense of self. This is where well structured learning environments come in, they can create the right atmosphere for outsiders to develop familiarity with these signifiers.

Research Question R2.)

What impact does CS10 have in attracting female students into the major?

From the findings in the quantitative analysis, we know that CS10 has a statistically positive effect on female students belief about CS achievement, and in particular, for female students

for whom CS10 is their first CS class. If we continue with our working hypothesis that belonging is a strong signifier for a students intention and ability to persist in the major, we can use the instruments constituting that dimension to analyze CS10's ability to attract female students into the major.

To do this, I am going to focus the attention of the data analysis on the experience of students who had previously attended CS10 and moved on to 61A. When participants were asked the question, "In this class I feel I belong?," The percentage of positive responses went down for female students. The number dropped from 51% to 27%. You can see this in figures 6.13c, 6.13d.

When comparing the experience of belonging between girls who had never taken any CS, with those of girls who had taken CS10 and then moved on to CS61A, this research found no statistical significance on their experiences. From figure 6.13e, you can see that the scores are almost the same for both cohort at around 27% of girls feeling a strong sense of belonging.

In answering R2.), i.e., the ability of belonging in CS10 to predict or say something about an intention to move forward in the introductory CS pipeline, I conducted a correlation analysis on the survey instruments that had to do with computational thinking efficacy, computer science efficacy, and belonging. The instruments of particular interest are the following:

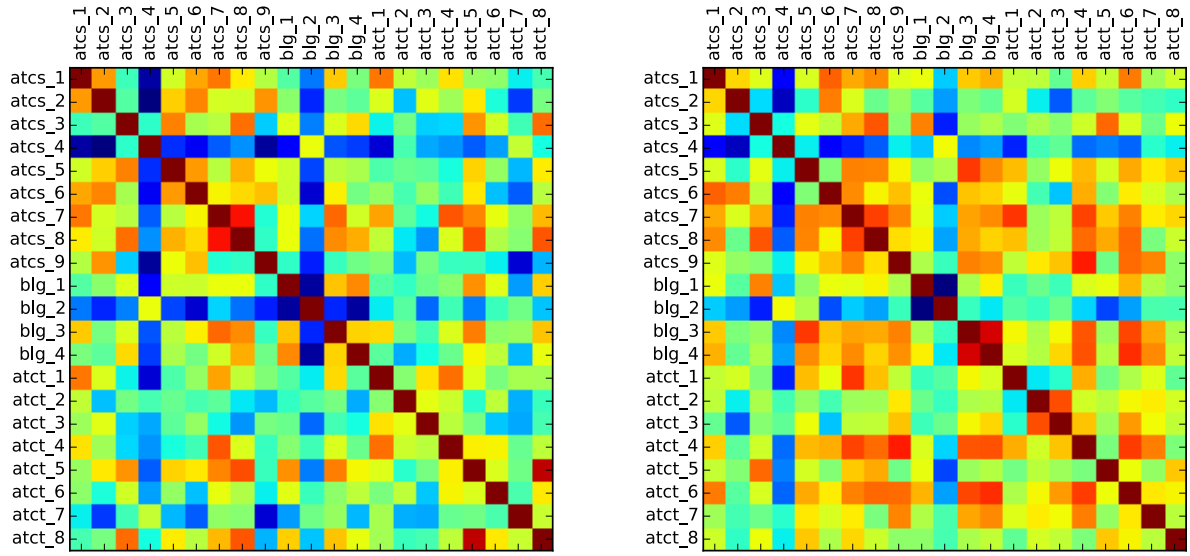
blg_1: In this class, I feel I belong.
blg_3: In this class, I feel like my ideas count.
blg_4: In this class, I feel like I matter.
atcs_3: I can achieve good grades (C or better) in computing courses.
atcs_8: I am confident about my abilities with regards to computer science.
atct_5: I know how to write computer programs.
atct_8: I know how to write a computer program to solve a problem.

For the cohort under analysis, i.e., the CS10 →CS61A female students, I found that for these female students, their self-reported programming ability predicts their sense of belonging, because the correlation heat-map from figure 7.1a showed that **blg_1** and **atct_5** are strongly correlated for girls with (r=0.60514), and **blg_3** and **atct_5** are strongly correlated (r=0.63538) [1].

For male students in the same CS10 →CS61A cohort, I found something entirely different. From the analysis and from the correlation heat map plotted in figure 7.1b, I observed that "**atct_4**: I am persistent at solving puzzles or logic problems," and "**atct_6**: I am good at building things," are strongly correlated with blg_3 and blg_4[2], with ($r \geq 0.7$) for most of the correlations.

[1] In interpreting correlations, for X and Y values between 0.5 and 0.9, the two variables are strongly related. That is, knowing X allows you to predict Y with considerably greater accuracy than if you didn't know Y. http://sites.stat.psu.edu/ jls/stat100/lectures/lec16.pdf

[2] blg_3 & atct_4, (r=0.70504)
blg_3 & atcs_5, (r=0.75434)
blg_3 & atct_6, (r=0.73270)



(a) Female (b) Male

Figure 7.1: Correlation Matrix of CS61A Students who had Previously Taken CS10

It is important to notice that usually for female CS10 students, based on the raw data spanning three semesters worth of experience, that **blg_1** and **atct_5** are *not* correlated, (r=0.31426), figure 7.2a. Since the relationship between these two variables seem to increase as girls move forward in the pipeline, as can be seen from figures 7.1 and 7.2b, this makes me believe this relationship is truly a strong predictor of CS belonging. Furthermore, in CS10, the data analysis revealed that female students sense of belonging in that class is highly correlated with their self-reported CS efficacy, as can be seen from the scores in table 7.1.

When one continues the analysis of the data to focus on the experience of CS10 males and females, as well as CS61A males and females, more insight is revealed on the self-reported attitudes of the students in the introductory to CS pipeline.

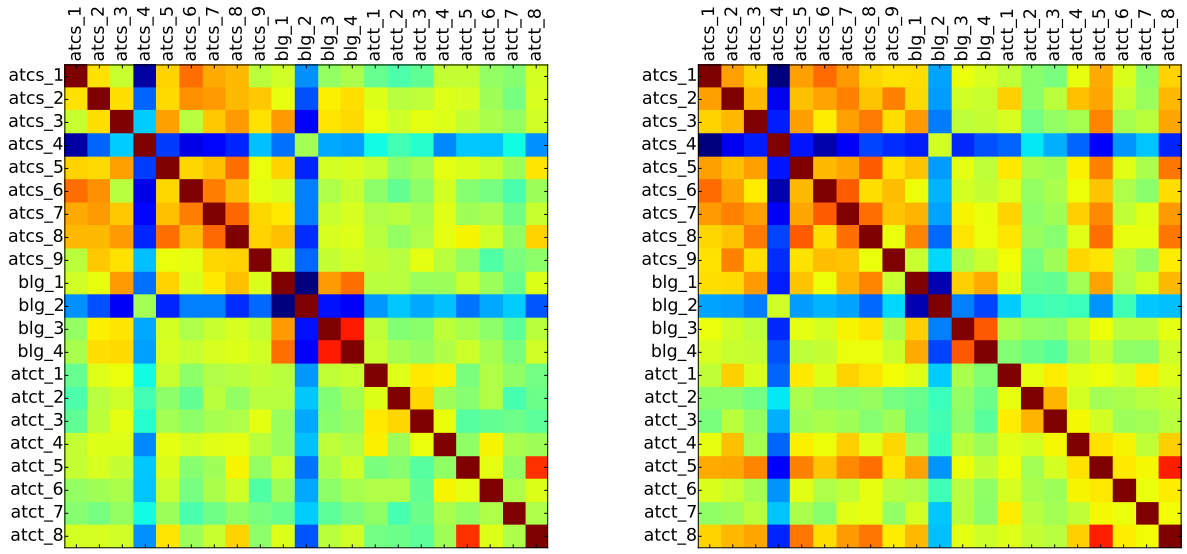
From figure 7.3a, we can see that the males in CS10 are the only subset in the study that do not have their sense of belonging correlated with their CS efficacy or their computational thinking (CT) efficacy, where as all the other subsets do as can be seen from figures 7.2a, 7.2b and 7.3b. However, once the gentlemen move on from CS10 to CS61A, from the data analysis, we find that their sense of belonging starts getting correlated with CS and CT efficacy, as can be seen from figure 7.1b.

blg_4 & atct_4, (r=0.71436)
blg_4 & atct_6, (r=0.77616)

Correlation with Belonging for CS10 Females

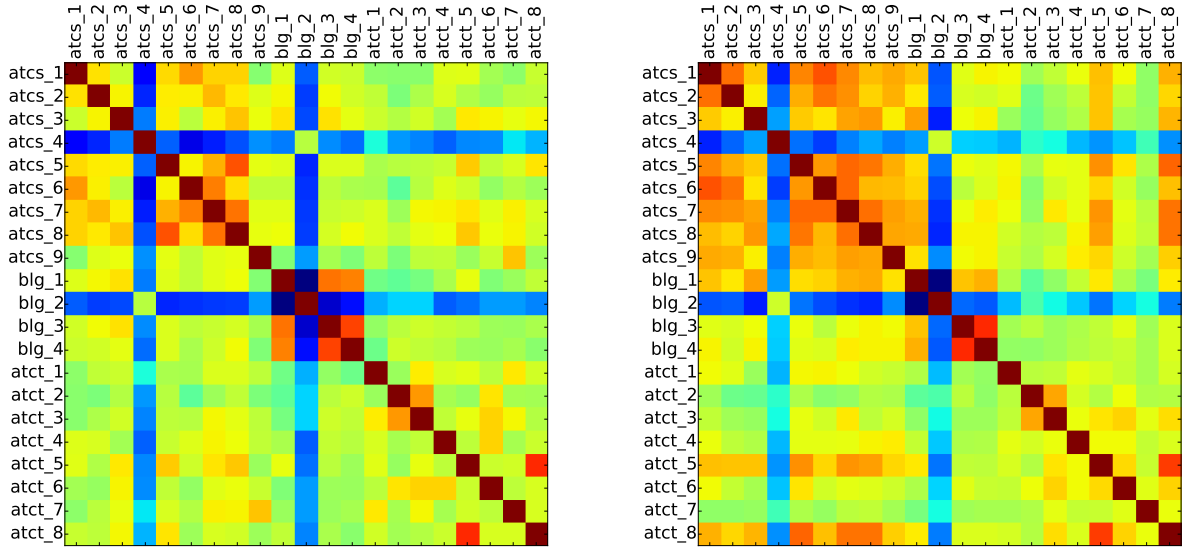
Instruments		Score
blg_1	atcs_3	0.56885
blg_1	atcs_8	0.50070
blg_1	blg_4	0.64547
atct_5	atct_8	0.75888

Table 7.1: Correlation Table



(a) CS10 (b) CS61A

Figure 7.2: Correlation Matrix Female



(a) CS10 (b) CS61A

Figure 7.3: Correlation Matrix Male

From this analysis, we can see that CS10 as great a class as it is, is not enough to overcome the major environmental challenges that the overwhelmingly male environment of CS61A poses to female students sense of belonging when they advance on to 61A.

Like I have said before, and will say again, the data analyzed for CS61A did not have the benefit of having female students who had taken the newly redesigned "Besides Block" module. From the analysis, we know that for the female students in CS61A, who moved on from CS10, their ability to program in Python has a strong positive relationship with their sense of belonging. The hope is that once students from this current iteration of CS10 move on to CS61A, their longer exposure to Python will be an added confidence booster in their belief of seeing themselves as CS people.

Furthermore, there is some good news to be had from the qualitative analysis with regards to answering this research question. Through my analysis, I realized that CS10 as bigger environmental effect than what I have discussed so far. For the males students who participate in the class, its seems to have socialized them positively towards working with women in a technical environment. One participant who I will call Ed, shared this experience with me, when our conversation ventured towards inclusion. When I asked him if he notice the lower number of women in the department, Ed answered as follows:

"Ed: Ha, that is also interesting, I don't, I don't notice. I would completely

> believe that it was, but it is not something I noticed. In 61B[3], out of the 10, 15 people I study with, probably 7 to 10 of them are girls. Most of the people in my study group are girls. So its not like I often feel like its male dominated, if I like took a step back I could completely believe that it would be."

It is important to note that when Ed took CS10, that was the first semester that there were more girls than guys.

Another interesting effect I found is that CS10s extended family—TAs, lab assistants, the readers[4], the BJCx[5] group— serves as an affinity group. This extremely diverse community inadvertently serves a retention mechanism.

One of the participants in the study, an Asian female student, with no prior CS experience, who took CS10 summer 2014, didn't feel like a minority at all in the department. Because the class was so mixed, even though she is in 61A now she is part of the CS10 family.

Research Question R3.)

What role does socio-culturally responsive curriculum play in attracting underrepresented students into computer science?

In CS10 learning environments, what happens is that instead of injecting *culture* into *education*, they instead, inject *education* into contemporary *culture*. They do this by bringing in conversation that investigates the role of computer science in the context of community, adhering to this Ladson-Billings paradigm which states:

> "In the classrooms of culturally relevant teachers, students are expected to engage the world and others critically" (Ladson-Billings, 1995).

For students in CS10, this line of thinking has become a salient part of their classroom experience, to the point of having extremely strong statistical significance, with ($p = 0$), with respect to their cohorts in CS61A, along both genders.

The social implications thread of CS10 has successfully had a statistically significant effect on student experience. Interpreting the findings along this dimension with regards to R3.), we find that introducing a social dimension in particular, seems important to the experience of female learners who have had some previous exposure to CS before they entered the undergraduate introductory to CS pipeline.

[3] CS61B, has a low representation of girls.

[4] A reader is a course grader

[5] BJCx, is the MOOC version of the BJC curriculum which is served through the University's relationship with EDx.

Additionally, through the data analysis, I discovered that for female students with some prior exposure to CS before CS10, the variable "**hiphop_d2:** Lyrical analysis gave me a new way to think about computation," was strongly correlated with their belief around CS success; "**atcs_3:** I can achieve good grades (C or better) in computing courses," ($r = 0.75942$). A similar relationship was found between the two variables, "**song_ct:** I was able to build on my pre-existing knowledge about songs to further understand computation," and "**atct_5:** I know how to write computer programs," ($r = 0.77585$); as can be seen from figure 7.4.

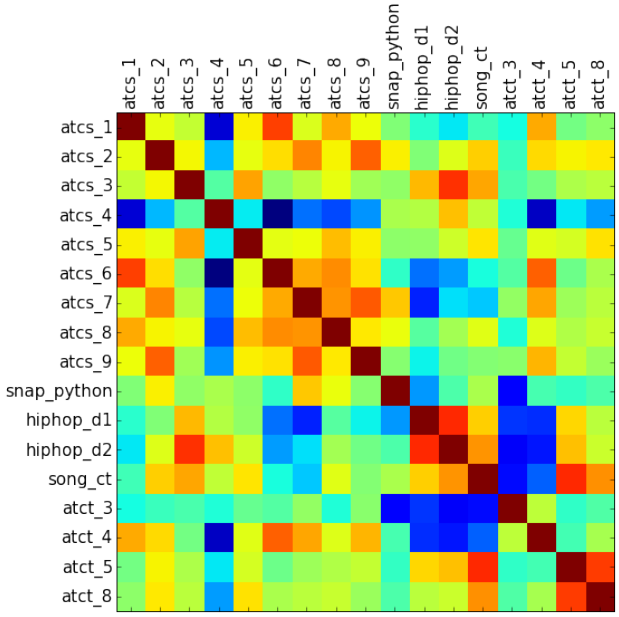


Figure 7.4: Correlation Matrix for Female CS10 Students with Prior Exposure to CS

From this finding, we can generalized to say that socio-cultural curriculum can have a positive effect in building female students confidence around CS.

How do we create Culturally Responsive Curriculum on Global Platforms?

If one accepts the answer that socio-cultural curriculum can be used effectively to equalize participation in CS, then the major challenge we now face as curriculum designers is scaling it for an increasingly global learner population. Better yet, how do we create a global frame work for culturally responsive learning that can be hyper localized in its instantiations?

The challenge for any CS MOOC is to tell the story of abstract, technical, material and ground it in a way that leverages cultural universals where appropriate and localization where needed. A core part of technical knowledge is gaining a clear understanding of the relationship between layers of abstraction. Even in the context of a small classroom, these abstract ideas can be rather difficult for many students to wrap their minds around, let alone

in an on-line course, where one does not enjoy the benefits of back and forth exchanges with students to clarify their misconceptions.

As educators, we understand the importance of using appropriate metaphors in knowledge transfer. From this research, we saw how building on student's pre-existing knowledge of songs was used to help them gain a better understanding of computation, which in turn has a positive relationship with learning how to write programs; illustrating why technical knowledge can be difficult to understand in the absence of accessible cultural mental models on which to hang knowledge.

An approach is to build in a way that is culturally responsive and adaptive by co-creating with a global developer community. Curriculum designers can ask developers to submit their suggestions of content that will help ground these technical concepts while leveraging their specific expertise of local lore. Once aggregated and produced, the appropriate localized content can be dynamically integrated into courses, based on students' geographical locations.

This gives an approach that both integrates open collaboration, with open education, leveraging social networks, in the service of building a globally scalable yet culturally responsive learning model.

Research Question R4.)

How does a graphical-based programming language affect students sense of CS identity?

Based on the working theory that students who have strong positive affect with regards to computational thinking, as measured by their scores on those dimensions are more apt to persist in the major, the data analysis of this study found that CS61A students seem to have a stronger programming confidence than students of CS10; even when one factors for gender.

Noting that a negative gender confidence gap exists in the experience of female students, what is *most* compelling about the findings in this study is that a female student who has taken CS10 has almost the same kind of confidence score about programming as a female student with no prior CS exposure who takes CS61A, as can be seen in figure 6.24.

CS10 is a programming heavy class, students spend all semester writing programs. With that understanding, and the additional knowledge that surveys were administered towards the end of the semester, each time they were administered, the only way I can justify these findings is by considering these two scenarios:

1. Perhaps students do not feel that programming in Snap*!* constitutes *real programming*.

or

2. Stereotype threat. Maybe the word "**programming**," itself is so strongly identified to the stereotypical idea of CS, that women and certain men subconsciously dis-identify with it.

Let's consider the first scenario. Maybe the reason why students answered negatively is that the word "programming" acts like an ambient cue, that primes negative stereotypes about the field that may cause a distancing with it.

Now, let's consider the second scenario. Most if not all students know that CS10 is a class designed specifically for non majors to explore CS. Furthermore, many student's know that Snap*!* and its variant Scratch are languages that are used by middle school and high school students. As a result, they may associate the language with an ability-demeaning stereotype since its not an industry language. This view was corroborated by themes that kept coming up in the process of the qualitative analysis. Speaking to a CS61A student about Snap*!*, he shared the following with me:

> "Boba: I am not an expert of the CS10 curriculum, but from what I have heard, like Snap*!* is kind of like a drag and drop language right, **I think its a little like maybe demeaning** for college students to have to work with programming language and programming experience that I would say middle schoolers or even elementary schoolers can interface with. Like dragging and dropping things instead of like they are a ***real*** computer scientist typing things into you know a text editor or whatever. So like its very surface level, I think it plays a very big role into how students perceive themselves. Because its a very big gap from like Snap to python which is used everywhere. And Snap which is used in CS10."

Another student whom I will call Lucas, shared this with me,

> "Lucas: With respect to CS10 I was naive about it ... Most importantly, the way that it looked, when you see things like dragging blocks around, it didn't look that good to me. I didn't think I had that much programming experience, but I thought I had good science experience."

This theme that drag-and-drop interface isn't "real" programming kept coming up, coupled with the low stature some of the students had for a programming language that is capable of accommodating pre-teens. Sometimes as academics, we forget that learning is deeply tied to emotions, if a learning context elicits a negative emotion, whether justified or not, students are not going to gravitate towards it.

About Snap*!*

With the low result of CS10 students' attitudes about programming, I will spend this section discussing the merits and challenges associated with the Snap*!* programming language.

Snap*!* is a programming language that was built exclusively as a didactic tool. One of the major reason why the language has such pedagogical affordances is as follows: When working in Snap*!*, the student is presented with a computational environment that literally makes the implicit mental models that we develop when we reason about computation explicit. This is achieved by creating a visual representation of the correct mental model; thereby, banishing misconceptions around those computational ideas. Whereas other languages were not built specifically for pedagogical affordance and rigor, they were built to build pieces of software for the end user.

Snap*!* was designed to help attain an *intuitive* understanding of computation; from building very simple things, to using computational primitives like recursion to build very complex things. With Snap*!*, the focus is on computation, it's primitives, the models that can be built from those primitives, and in gaining an understanding of all the layers of abstraction that emerge as a function of combining those computational primitives in evermore purposeful ways.

Whereas, in other programming languages, the focus isn't on the computation that is realized in that language, instead, the focus is on the *artifact* that is created through the use of that language. It's a shift in cognition from the tool to the end product of the tool.

During my interviews, I spoke with a CS10 lab assistant that I have named Saturn, to get insights on how she sees the language, and how she observes students work with the language. Saturn shared the following:

> "Saturn: Snap*!* is interesting. It works really well. I think it gets cumbersome pretty fast, which is why we teach in Python, things are just faster if you can just type versus these laggy drag and drop.
>
> But Snap*!* does provide, I think the coolest thing is the stage with the sprites, all lot of projects involving visual things, you are not going to be able to get that visual polish with Python.
>
> I think it is helpful in the beginning, it helps to enforce syntax correctness on you, because it just won't allow you to do certain things. Its definitely as stepping stone."

From Saturn's remarks, one sees that she gets the raison d'être of Snap*!*, but she also has an appreciation of the challenges it poses in terms of its ability to be nimble. Another participant, Bean, shares his view on Snap*!*

> "Bean: [hesitates] You know I liked it, in the beginning because it was very user-friendly and it's a lot of, it's less intimidating when you have the blocks on the side like building blocks, whereas in Python, you open up a text editor and there's nothing there and you're just, you know, making everything on your own.

> So that was really nice but at some point during the semester, I felt a little limited with Snap*!* Especially the most annoying thing for me was when you had great projects and there is something you have to edit in Snap*!*, and the drag-and-drop just really isn't conducive to editing things that are like nested for loops and things like that, it's hard to get a handle on those kind of things."

Bean especially nails it. His insights helps to shed light on the appropriate role Snap*!* plays in the learning experience of the students. Snap*!* is a great language to on-board students into computation and not necessarily computer science. Its genus as a pedagogical language has the unintended consequence of hindering students' sense of CS identity. For the student whose CS identity is already forged, Snap*!*'s strength as an aid in understanding computation can be truly appreciated.

My recommendation is that Snap*!* should be used as a supplementary language that is brought in to help clarify some challenging computational ideas. For example, when we are introducing a topic like "higher order functions," Snap*!* can be brought out to aid in instruction because Snap*!*'s full power, is in its ability to lay bare complex computational ideas in simple visual ways that communicate appropriate mental models. This helps learners grasp and understand complexity in mind-sized bites. That is the stunning power of Snap*!*

Yet from the data gathered in this study, it is clear that the strength of Snap*!* is not understood and respected by most students. We are near Silicon Valley, where a student can come to UC Berkeley and have a year of computer science under their belt and go into an internship at a top high-tech company, where they are paid the equivalent of some people's yearly salary for three months of work. For students, that is a really compelling reason to not revere Snap*!* and not feel confident about their programming skills. Whereas when they spend their time using an industry ready language like Python, students feel much more prepared, and much more confident that they are doing *real* programming.

Based on this finding, one can posit that the extensive use of Snap*!* might be moving computer science pedagogy away from being culturally relevant. The prevailing culture in computer science, and specifically the prevailing culture in Silicon Valley is one in which the professional programmer is the new "rock star." And it is not an unfair assessment to say that students find this idea seductive. Not only that, professional software engineering is also a solid way of earning substantial income.

What we have here is very similar to what Ladson-Billings has noticed when approaches counter to the notion of cultural relevancy are applied to populations. I will take a section from one of her papers to illustrate a point.

> "During the 1960s when African Americans were fighting for civil rights, one of the primary battlefronts was the classroom (Morris, 1984). Despite the federal government's failed attempts at adult literacy in the South, civil rights workers

> such as Septima Clark and Esau Jenkins (Brown, 1990) were able to teach successfully those same adults by ensuring that the students learned that which was most meaningful to them. This approach is similar to that advocated by noted critical pedagogue Paulo Freire (1970)."(Ladson-Billings, 1995)

From this excerpt, one can conclude that the government's failed attempt at adult literacy came because the government teachers were teaching a curriculum that they deemed important, but whose importance wasn't recognized by the adult student population. Taking a cue from Freire's stance on education for liberation, the civil rights leaders were able to achieve what the government teachers could not. They listened to their students and gave the students what they needed.

According to Freire and his pedagogy of the oppressed, we as teachers and curriculum designers need to give students the skill sets that make them feel empowered (Freire, 2000). Furthermore, the students' need for liberation through their education should supersede the need of the teacher or the curriculum designer, or the need of the institution in which the learning occurs.

It would seem that we the curriculum designers are solving a different problem than that of our students. Our students are more interested in going off to industry internships as soon as they have some skills under their belt, and we are more concerned with them mastering the art of computation.

At the end of the day, I do not blame the students when Silicon Valley glorifies people who have mastered the nuances of programming languages instead of the complexities of problem solving. Furthermore, as co-creators of education with our students, and someone who believes staunchly in education for liberation, I believe its time that we start working with our students and not against them.

Besides Blocks

This finding further supports the need for a transitional module helping students navigate programming from a graphical language to a text-based language. The good news however is that these data say nothing about the effect of the "Besides Block" module; which was introduce at the same time in CS10 as when the data from CS61A was being gathered. As such, the students in CS61A, who had previously taken CS10, did not have the benefit of taking the "Besides Block" modules.

Because of the my research timeline, I was able to interview some students who had taken the Besides Block module and had since moved on to CS61A, to evaluate the effect of that intervention. I interviewed a student who shared with me how she was able to successfully transfer her knowledge from Snap*!* to Python. This particular student did something very similar to what I had seen in my observations at EPGY, the summer I taught middle

school CS at Stanford University, which ultimately inspired me to suggest the addition of a transitional Python module to the BJC curriculum (chapter 1.5).

> " Interviewer: What has been your experience in going from writing code purely in Snap*!* to now writing code purely in Python?
>
> Willow: It was not too hard switching from Snap*!* to Python because there are lots of similar aspects and stuff. It was a little tricky at first getting the syntax down because Snap*!* does have slightly different syntax from Python, but learning any computer language, they're all going to be differences.
>
> Certain things I guess were tough, like when we came to higher order functions and stuff like that because we learned it as one thing in CS10, and we learned it as completely something different CS61A, that was kind of tough getting that down. But normally from code to code, line to line, it really was not that difficult, especially because it was nice to kind of like think about how to do this in Snap*!* and just figuring out the words for it in Python.
>
> Interviewer: In any of your code in Python, did you at any point switch to Snap*!* to figure out what you wanted to do, and then transferred over to Python?
>
> Willow: Yes I did that especially in CS10, I don't do it as much in 61A because I have got it down, within like the month of learning python in CS10, but sometimes I would have like Snap*!* open and look through all the things that I could do, it's nice because Snap*!* has a word bank for you, whereas Python, it's just like what is in your head, so I definitely looked at that especially during some of the labs in CS10, there was like Snap*!* right next to Python code so you could just transfer it over. I definitely did that at some point, and even now, CS61A, we are learning a new language called scheme, I am writing things in Python so that I can transfer it into scheme now."

From a learning perspective, this student implicitly understood that the computation she had learned was the same. All that she needed to figure out was how to do it in the new medium. Not only was she able to learn Python better, she then used the same learning technique to transfer her knowledge of computation from Python to Scheme.

Another student I spoke with, this time a lab assistant shared with me how students were embracing the modules. Pluto states,

> "Pluto: I took CS10 summer 2014. I am a lab assistant and in 61A now. A lot of students actually use Python to do their final projects this semester. Last semester we had like one or two people do Python, this semester I've already seen five or seven projects in Python. They want to see the terminal because that's like everything they see in the movies, or they want to see sublime and stuff like that."

This particular lab assistant was there fall 2014, as well as spring 2015, so she had a good grasp of how things were going in CS10 labs. Her comments supported my intuition that students want to emulate what they see in the media when it comes to CS, putting them in that context, where they are using Sublime Text[6], pulling up the terminal, and typing in a few UNIX commands. These acts make them feel that much more confident that they are transitioning into "real" programming.

From the results of the analysis on the Besides Block modules, one can conclude that it was successful in acting as a "transitional" object that allowed students to transfer their knowledge of computational thinking from Snap*!* to Python. This showed that the labs helped students decouple their understanding of computational thinking from Snap*!* programming.

Data Module

In his work on African Fractal's, Eglash notes:

> "It is not as if we claim that this is a more African way to learn about fractals. Rather, we attribute this enhancement to improved opportunities for conscious reflection about heritage and culture in relation to computing and math, and improved access to an analogous computational reflection or tinkering." (Eglash et al., 2011)

While Eglash's ethno-computing mathematical simulations had raised students scores by creating a computational task that allowed them to naturally reflect on the relationship between computation and culture. In designing the data module I wanted to test a similar affect. I wanted to know if it *too* enhanced how students thought of computation.

Like I shared earlier in answering R3.), I discovered that for female students with some prior exposure to CS before CS10, the variable "**hiphop_d2:** Lyrical analysis gave me a new way to think about computation," was strongly correlated with their belief around CS success; "**atcs_3:** I can achieve good grades (C or better) in computing courses," ($r = 0.75942$).

Similarly a relationship was found between the two variables, "**song_ct:** I was able to build on my pre-existing knowledge about songs to further understand computation," and "**atct_5:** I know how to write computer programs," ($r = 0.77585$); as can be seen from figure 7.4.

Why is it that some aspects of the data module correlated with some female students' belief about their ability to do well in CS classes and their ability to write programs?

Here is what I think is happening. These female students had already had a prior exposure to the field, they had a rough idea what CS was. By scaffolding a connection with culture, switching the context of computation from an abstract concept to something concrete they were already familiar with probably made them feel that much more confident about this

[6]Sublime Text is an industry standard text editor.

path. Furthermore, from the findings in this study, as female students move forward in the CS pipeline, their sense of belonging as measured by **blg_1** becomes more correlated with their belief in their own programming ability.

With respect to its emphasis on culture in relation to CS, the data module did perform as was intended for a subset of the study population. I was slightly disappointed to discover that the questions around computation and culture with regards to this module had not had an impact on the other subsets of students.

One reason for this could be that the demographic of students in the study, i.e., mostly White and Asian, may not have had a culturally salient connection to Hip-Hop. As a result, it may not have resonated with them as I think it would have for students from whose culture Hip-Hop emerged. This hunch was somewhat strengthened from my interview with Rapunzel.

Rapunzel had a lot of challenges in CS10. First, she is a transfer student who was specifically ***warned*** against taking any CS classes by her friends and advisors[7]. When I interviewed her about her experience with the data lab, she states:

> "It was the one they were talking about going through lyrics and sort of demonstrating why, basically how to go through big sets of data, right? Let me see, this was quite a while ago, I am not very sure, I do remember the lab though. I felt that it was, let me see, [takes a minute to go to the lab on their computer], **that definitely says something in itself that I am able to remember what this lab was because I wouldn't even know um what any of the other ones were.** I dont believe I was able to get it checked off but I did actually do it, I have the answer and stuff prepared for it.
>
> Because I did recognize that it was, it seemed like this was the first lab to sort of incorporate like how you would use this information in real life, because I can imagine like somebody actually needing to go through possibly a list of lyrics for say a Decal or a course or something like that, but I wouldn't be able to see somebody actually doing something like fractals or recursion or something like that, I don't see how those thing would be useful in real life outside computer science basically.
>
> But, also very much the lab as an attempt to bring culture into it, it also seemed that it was intentionally put there, because it sort of stood out, it was kinda odd because it was the only one."

[7]NB: Interviewing Rapunzel was a very painful experience for me. Most of the things she shared made my heart sink. This interview tested my professional resolve. The professional detachment I have to put on in order to not bias the interviewee. I am certain if I wasn't a Nigerian American female, this story would have never come out. She would have probably not felt comfortable enough to share this experience.

Two things are happening here. Rapunzel who had had a difficult semester in the class, specifically remembered that lab. It left an effect on her. She could see for the first time, how something she was learning in the class could be applied to a context that she was familiar with. Remember, we learned about Rapunzel earlier and how she represents that very important demographic, that of the "complete outsider." Nothing in her experience before Berkeley had prepared her for the world she now found herself in. When I asked Rapunzel if she had any prior exposure to CS before Berkeley this is what she said,

> "No, not at all!
>
> Not even a little bit.
>
> I have no idea what it entailed.
>
> There were other courses that did fulfill my quantitative requirements but, my friends had taken courses such as statistics at community colleges or some other such course. I have a terrible experience with community colleges where they would not transfer my work or something like that so I knew that I have to take a course at Berkeley even though I was strongly advised against it.
>
> So I wanted to steer clear of mathematic courses and statistics courses. So I thought the very basics in computer science would be very good both to steer clear off that and to fulfill my quantitative requirements."

I was taken aback when she told me that she was warned against taking CS courses. It seemed like the opposite of what we as the designers would want. Here we are designing courses specifically to equalize participation in CS, and there they are, whoever *they* are, advising students not to try the field. Implicitly reinforcing the notion that only certain people do well in CS, and those certain people don't look like her. When I asked Rapunzel about this, this is what she shared:

> "Oh my gosh every body, [Laughs] okay let's see I was advised against it from friends I had met. I lived on the African American floor my first semester here, and a lot of my friends now remained people that I met during that time.
>
> So every time that I mention that I wanted to take it they suggested that I take a course in community college which I didn't think I will be able to do because of my prior history with community colleges.
>
> I was also advised against it by the people in the ELP program, they know that it is difficult for people who have not had any kind of exposure, and those who have taken it who have given accounts to them have told them how hard and strenuous it was, even though ELP counselor knew my circumstances, they know how things have been difficult for me from my years here.

> I really should have listened to them and to let it fall down to my final moments here, and I needed to get something done, so I settled for computer science.
>
> Also I felt like perhaps it might be the case that I might use it later, but now I am not very sure I thought it might have been helpful, you know."

The story of Rapunzel exemplifies the promise that socially responsive curriculum can bring. Helping an outsider find some ground of familiarity to stand on while learning computation. The more we can do this, the better. Where appropriate, we can bring in Snap*!* and help ground our teachings even more. But we must be smart about how we apply our pedagogical languages so that in our intention to help our students, we don't hurt them by giving signifiers that only serve to highlight their outsider status even more.

7.2 Conclusion

Often in academic discourse, we talk about the problems of minority students, or the problems of female students in computer science. When we are saying this, we are often not even aware of what we are saying. If you analyze a statement like "the problem with minority students in computer science." Literally what we are saying is that minority students *are* the problem. We have stripped them of their humanity. When in fact, the way we should be discussing this issue is to say correctly, "the problems that students face" with computer science. Black people are not the problem, they are first people, and then people who *face* a social problem.

The norm in CS as it now is mostly that of a White and Asian male. Often times these students have challenges in sharing spaces with others. Historically, our society has been mostly de-facto segregated along race as well as gender. Most of our learners grew up with ideas about the appropriate roles for women, for men, for White males, for Asian males, for Brown women and so on. It is collectively our fault because we have failed to socialize them well in terms of working, living, respecting and collaborating with people who are not White or Asian males. We as their parents, we as their educators, we as their leaders, we have failed them.

We need to move away from diversity and move ever closer towards inclusion. Diversity and inclusion are not the same thing. Diversity alone will not solve any problem. This is often why well-meaning diversity programs fail. The presence of historically underrepresented persons in the CS milieu is not enough to make such an environment feel warm and inviting. Yes their presence needs to be there but that in itself is not enough. They need to be fully integrated into the social fabric of the environment to feel a sense of belonging. In essence for historically underrepresented learners to feel a sense of belonging in CS, hey need to transition from outsiders to insiders.

What one often finds when these kinds of programs are enacted, is a resentment on the part of the person that has already been there. This is why sometimes African-American

students get pilloried and labeled as "affirmative action tokens," because majority of students see them as not earning their way into those positions. Whether that is right or wrong is completely irrelevant to this discussion.

What is of relevance is that sometimes diversity programs create a confrontational framework which ends up creating an environment that is charged and full of repressed emotional resentment often on both sides.

7.3 Recommendation

Broadening Participation with a Focus on Inclusion

An approach at solving the problem that may arise from well-meaning diversity programs is to take a page out of the tried and true *Treisman* approach (Treisman, 1992) and de-politicize the whole enterprise. Our focus as practitioners should be on getting problems solved. We should not concern ourselves with winning political correctness politics.

The findings of this study revealed that many students had been introduced to the field by a family member, many others grew up in Silicon Valley where their parents were software engineers, or their neighbors were, or their friends parents were.

We can't move the whole world to Silicon Valley, and give people the privilege of geography. But we can leverage something very interesting that came up in the course of this research, i.e., the revelation that many students decided to major in computer science once they discovered for themselves how truly collaborative the field really is. This aspect of CS can be highlighted in campaigns, when we leverage media to change the perception of CS. When these students show up in our classrooms, we should embrace them all with open arms, remembering that everyone is capable of learning CS.

CS Strength is its Weakness

For those that love CS, it can be all consuming. Computation is extremely fascinating and addictive, especially when adherents are trying to solve some of the computational puzzles that arise through the work. Problem solving for those that love it, gets you quickly in the state of flow, and by the very nature of its complexity and the belief that if you just keep at it, you will eventually unravel the puzzle. Just like with video games, once a computer scientist unravels a computational puzzle, this leaves the computer scientist in a state of euphoria, which reinforces the reward centers in the brain, that keeps the scientist coming back for more.

Ironically, one of the best aspects of computer science is what is actually working against this issue. CS is extremely collaborative. It processes are so collaborative to the point that the collaboration quickly evolves into a form of sharing life together. This collaborative experience easily spills beyond the boundaries of a 9-to-5 work life.

Integration Instead of Assimilation

If computer science is indeed a way of life, and historically, the culture of that way of life has been created by mostly White and Asian males, the question we have to ask is "what this means for people who are neither White nor Asian males?" For the none male White and Asian student, being successful in CS often means assimilating into a male White and Asian norm. For the female student, this might mean eschewing of dresses and skirts, and the adoption of a *brogrammer* uniform of hoodie, and tee shirts emblazoned with technology company logos. For the Black student it might mean an the acceptance of spending majority of their social time with their White and Asian colleagues with little time to spend on activities that are culturally Black.

For those of us who seek to equalize participation in CS, the challenge the creation of learning and living environments that allows for learners to bring all of themselves to the computer science way of life, without diminishing or reducing who they are culturally.

The challenge for CS as a field is the design challenge that allows *true* integration to happen versus assimilation. In CS, a similar thing is happening to the dis-identification with schooling that scholars noticed with African American youths. In CS, we see intellectual capable students disengage with the field out of a subconscious fear of complete assimilation into a male White and Asian norm. The fear that one will be erased, is one of the reasons that leads people to recoil from the CS.

What we need to equalize participation in computer science and create environments where ethnic-minority people will thrive, is to train people in *cultural fluency*. We need an environment that is truly *integrated*, and not one in which everybody comes in and gets whitewashed into the existing norm.

In the following section, I paint a scenario where CS is enriched by our diversity, which in turns shapes the culture for inclusion. I envision a CS environment where an European man comes in and brings with him his European culture and crafts his experience of computer science to support his culture. Similarly, I envision a CS environment where an Egyptian woman enters the field and brings with her, her culture which she integrates into the field. In an inclusive CS environment, she would not need to reduce or diminish any aspect of who she is. The same thing would happen for the Chinese girl, the Nigerian boy and so on.

No CS Ghettos!

For the scenario that was previously painted to happen we have to have good enough representation of all groups. What ends up happening when attempts are made at addressing this problem is the creation of cultural ghettos. We end up designing environments based on our subconscious notions of supremacy and exclusivity. We create the case where we say, "This is the school that caters to the Black computer scientist." We fall back on solutions that are separate but equal.

Those solutions are blatantly unAmerican and says the issues of race and culture are so impossible, so intractable, that we are not going to even bother with them. We are going to dispense with the fact that the races cannot successfully co-create and co-habitate in the same space. I blatantly refuse to believe such a notion. Along the same lines of thinking, we proffer solution that create a gender ghetto were students are ushered into single-sex environments for computer science.

The solution is inclusion! The solution is an inclusive framework that is very difficult for us to understand in the context of our American lives. As a nation, we tout "diversity," but fail to understand that diversity without inclusion doesn't lead to the outcome that we want. In order to realize the importance of inclusion, we have to change the mental model that we use to think about issues of equalizing participation. We are *conceptually*, a supremacist society. Most of us grew up in a world that was conceptually monotheistic. Where there were no cultural frameworks set up to help us understand inclusion.

If we grew up in a society where for example, it was polytheistic, then we would have at least seen a pluralistic framework, that we imparted excellence in. Such a framework could give us a mental model to understand inclusion as a collective good. Instead we have a cultural framework that sees difference as deviance. This is the real challenge that faces equalizing participation in computer science; the challenge of the evolution of thought from "supremacy" to "inclusion."

A lot of what this study has uncovered cannot really be effectively corrected in traditional institutions for several reasons. Sometimes the DNA of such institutions are such that they rejects any attempt at redefining their culture. Other times, these institutions have become too big and too politicized.

Recreate Engineering Culture: Boot-Camps

The solution is the creation of another model for delivering computer science education at scale. A model that is not beholden to the old archetype of the University system. I believe that kind of institution already exists in the "coding boot-camps." When these institutions emerged, many scoffed at them. The critics said they were using market forces to address education, and therefore, the injection of capitalism into education could only lead to failure. Nevertheless, coding bootcamps present an interesting opportunity to reinvent engineering culture from the ground up.

What if this is our chance to recreate engineering culture from scratch? What would this new culture look like? Its begs us to ask the question, "What does a good engineering culture have?" It would have to help people invent themselves as compassionate engineers with rigorous socio-technical training. It would have to engender in the learners an understanding of engineering as both an art as well as a science.

What if learners gave us their time, brought with them their passion, and we in turn help transform them from being CS outsiders to being CS insider? We CAN do all these things if

we invent a learning environment that allows us to freely serve the needs of learners without having to accommodate the needs of research institutions.

I believe the craft of software engineering should be in service to the world. The world is messy and filled with social issues that make us feel overwhelmed at times. This is where we as learning scientists, inventing new learning institutions, come in. We can coach and encourage learners to engage in courageous conversations. Software doesn't exist in a vacuum, and software engineering cultures should not exist in one either. Software engineering cultures should exist fully embedded in messy society.

If we refuse to engage in challenging social discuss, then we will not fulfill the mission of using our craft to invent a compassionate future. The solutions that are posited here, the solutions that we experiment with ultimately has to scale globally because the issue of equalizing participation in computer science is not just an American problem.

7.4 Summary

This dissertation had laid out the challenge of equalizing participation in computer science. A proposed solution of designing a new APCS Principles course with the explicit goals of broadening participation was presented.

A mixed-methods formative research was conducted that sought to to answer the question, "What are the socio-curricular factors that lead historically underrepresented students to choose CS?" In answering that question, an exploration of related scholarly work was presented which had to do with computational thinking, the barriers to learning and the complex nature of underrepresentation in computer science.

In answering the main research questions concerning this dissertation, an in-depth examination of the UC Berkeley course titled The Beauty and Joy of computing was presented juxtaposed in its analysis as a means of broadening participation against the UC Berkeley course titled The Structure and Interpretation of Computer Programs. The design and theoretical rationale for the creation of a transitional learning module called Besides Blocks, which reaches its apotheosis in the Python Data lab was presented. Findings from the formative mixed-method research which demonstrated the efficacy of socio-cultural, responsive, curriculum in broadening participation in computer science was put forth.

My hope is that as we like-minded-people continue to work in this area of equalizing participation in computer science, maybe she can finally be the radical, innovative field she has always said she was—by bringing about a different way of life for the 21st century. If CS as a field is able to successfully solve the issue of integration versus assimilation, and she is able to successfully bring in women into her field at a global level, that would have been one of the most important things to have happened in this 21st century. When you educate women and give them a living wage, they will have less children, and those children will go on and have a higher quality of life; which will reduce the amount of human beings on

an already straining planet. Maybe, just maybe, this is the moon-shot idea that will solve global inequality.

Appendix A

Hiphopathy Lab

A.1 Introduction to Data Science using Hip-Hop Lyrics

Welcome to an innovative way of teaching and sharing the delights that comes when data, computational methods and culture collide. The goal of this lab is to connect cultural relevance to computing by introducing elementary techniques of natural language processing with a corpus of hip-hop data.

Art is something that we often believe that is separate from the so-called hard sciences. As a result, those of people who are gifted in the arts or are inclined in that direction, shy away from the sciences because many fall prey to the false choice of choosing between the arts or the sciences.

We want to change this false dichotomy by bringing the power of computational methods to investigate and explore the riches that are inherent in the arts. It is for this reason that we have chosen to use a corpus of hip-hop lyrics as the dataset on which we will practice our computational techniques. At the end of this lab you will be able to build a visualization of the occurrence of certain words in a rap corpus.

One of the reasons why Python is loved is that it is a language that lends itself well to things like building webapps as well as "hardcore" computational science like calculating the orbits of planets and stuff like that. What we are interested in, is doing a bit of data manipulation based on a hip-hop corpus. We are going to be doing something similar to what Matt Daniels did with the hip-hop vocabulary. We are going to get under the hood and create our own hypothesis.

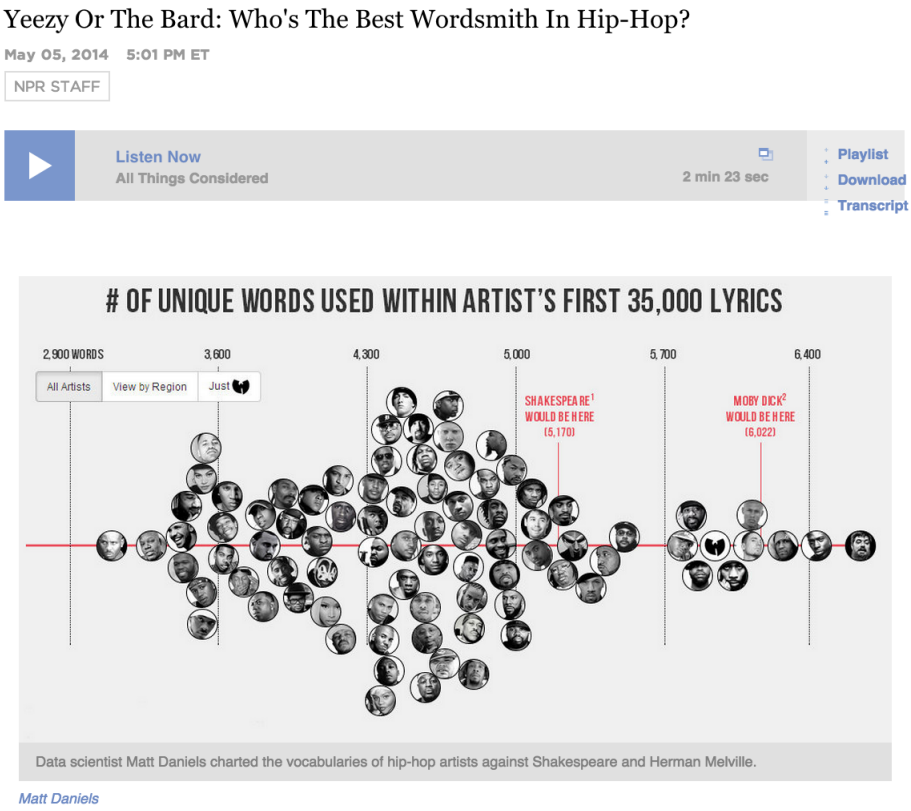


Figure A.1: Yeezy Or The Bard: Who's The Best Wordsmith In Hip-Hop? - NPR Music Story

A.2 The Natural Language ToolKit

This section provides information regarding how to run `nltk`. The computers in the lab do not have `ntlk` and its accompanying libraries installed. The easy way to do go this lab is to either use the Python data analysis browser application of wakari.io, or you can download the anaconda program, which installs a full scientific stack of Python on a mac.

If you read the article, you will see that Matt used the first 35,000 lyrics of each artist. For the sake of simplicity, I am going to use the artist Jay Z as the subject of our analysis. So let's go and collect the first 35,000 words of the Jay Z lyrical catalog.

How are we going to do this you might ask? Well first off, you can go to your favorite search engine and search for Jay Z lyrics. On the other hand, you can actually use the rap annotation site genius(http://rap.genius.com/) to get all that information and then some. According to Genius, Jay Z has a lot of songs. On average, most rap songs are usually set

with 3 verses containing sixteen bars, or sixteen sentences each.

```
>> 35000/(16 * 3)
729
```

So if 16 by 3 = 48, and 48 goes into 35000 gives approximately 729 songs. That can't be right, even though Jay Z is prolific, he hasn't written 700+ songs. So I must have gotten my understanding wrong.

I proceeded under the false assumption that each lyric was a sentence. Now I can see that each lyric must mean each word instead. This brings me to an essential quality of a good problem solver and Computer Scientist, the ability to embrace failure. More on that later. So lets go back and re-analyze the numbers. I need to figure out how many words are in the average rap bar?

In order to solve this we need to basically find every instance of Jay Z's lyric, scrape them off the Internet, and then start number crunching on them. To make this process faster, I have already built something called a webscraper and scraped all his lyrics till the Holy Grail album. Hopefully that gives us more than enough data to get to 35,000 words.

Before we go any further, let's make a directory that will hold our Python data files. We can enter the command `mkdir PythonDataLab` in our terminal to create a directory called `PythonDataLab`. Now that we have made this new directory enter the command `cd PythonDataLab` to go to the directory.

The lyrics I scraped off the Internet have been compressed in the zip file named 'JayZ.zip.' Go ahead an unpack this file to our PythonDataLab folder. Take a look inside, you will see its made up of a bunch of text files. Take sometime and go through some the text file to see what they look like. For example the file "JayZ_The Black Album_99 Problems.txt" contains the lyrics to 99 problems.

Now that we have these files, we are going to use some Python packages (a package is also known as a library) to help us. The Python natural language toolkit (NLTK) is one of the more popular Python libraries that people use for natural language processing. In fact, Matt Daniels used it for his hip-hop vocabulary. You can learn more about NLTK here, http://www.nltk.org/book/.

So lets import the toolkit. Go ahead a fire up Python in terminal. To import any package in Python, you type in the keyword **`import`**, followed by the name of the toolkit.

```
>> import nltk
```

The nltk library has a lot of functions associated with it. The one we are going to use in particular is called a Corpus reader. If you look up the definition of a corpus, you will see

that it is just a collection of written text. The JayZ folder contains a corpora of Jay Z lyrics. The great thing about nltk is that it comes with built-in support for dozens of corpora. For example, NLTK includes a small selection of texts from Project Gutenberg, which contains some 25,000 free electronic books. To see some of these books we can run the following query

```
>> nltk.corpus.gutenberg.fileids()

['austen-emma.txt',
'austen-persuasion.txt',
'austen-sense.txt',
'bible-kjv.txt',
'blake-poems.txt',
'bryant-stories.txt',
'burgess-busterbrown.txt',
'carroll-alice.txt',
'chesterton-ball.txt',
'chesterton-brown.txt',
'chesterton-thursday.txt',
'edgeworth-parents.txt',
'melville-moby_dick.txt',
'milton-paradise.txt',
'shakespeare-caesar.txt',
'shakespeare-hamlet.txt',
'shakespeare-macbeth.txt',
'whitman-leaves.txt']
```

Notice that in order for us to use the functions associated with `nltk` package, we have to put the `nameofpackage` followed a '.' dot operator, then the name of the function we need. This is the usual Python formalism. `NameOfPackage.function`

Ah, there is Shakespeare's *Macbeth*, *Hamlet*, as well as *Julius Cesar*. There is also the King James version of the bible as well as Jane Austen's Emma. Let's get on to the business of making our JayZ corpus.

Depending on whether ntlk books have been downloaded before on the computer you are on, your query may not successfully complete. If your query doesn't give you some of these books, enter the command `nltk.download()` This will open a window, or show a command list in your Python environment. Go ahead and download the corpora if you would like to play around with it. Otherwise, feel free to skip this, as we will build our own corpora.

A.3 Literary Corpus

Now it's time to seriously write some code. Start by right clicking on this link and select save as. Make sure to save this file to the PythonDataLab directory. Head back to the shell and enter the ls command to make sure that the file made it into our PythonDataLab directory.

```
>> from nltk.corpus import PlaintextCorpusReader
>> corpus_root = 'JayZ'
>> wordlist = PlaintextCorpusReader(corpus_root, '.*')
```

Based on my perusal of the nltk book, I know that there is a plaintext corpus reading function named `PlaintextCorpusReader` that I can use to make my corpus. From nltk corpus function, which I access through the '.' (dot) operator, I import `PlaintextCorpusReader` from `nltk.corpus`. I create a variable that I name `corpus_root` and assign the folder.

```
corpus_root = 'JayZ'
```

I then call the plain text corpus reader function with the root location and the token "`'.*'`" that means grab every file in that folder.

```
wordlist = PlaintextCorpusReader(corpus_root, '.*')
```

I have adapted a function from nltk to reading in my corpus, `create_corpus`. The definition for this function can be found in the lyric_analysis.py file. In order to make this function work, we will need to get some additional libraries of functions. In this case, we will need the regular expressions library named "`re`". So the first thing we need to do is import that package.

```
>> import re
>> the_corpus = create_corpus(wordlist, [])
```

Now `the_corpus` contains all the lyrics. I wrote the `create_corpus` function in a way that shows what lyrics were read in. You should have a similar output to the one below.

```
>> the_corpus = create_corpus(wordlist, [])
JayZ_American Gangster_American Dreamin.txt
JayZ_American Gangster_American Gangster.txt
JayZ_American Gangster_Blue Magic.txt
JayZ_American Gangster_Fallin.txt
JayZ_American Gangster_Hello Brooklyn 20.txt
JayZ_American Gangster_I Know.txt
...
```

The series of words that make up Jay Z's lyrics is now represented by a list data structure, named `the_corpus`. This list data structure is the same exact computational mental model that we have already acquired with the list data structure we are already familiar with in

Snap!. Hopefully, you are beginning to gain a better understanding of how all the computational thinking skills you acquired in your learning of Snap! carries over to solving any computation problem realized in any programming language.

A.4 Frequency Analysis

We have finally gotten our Jay Z Corpus! The data that we collected has a 112,871 words. You can see for yourself by running the **len** (Python function for figuring out the length of a list) on `the_corpus`. This is great, because we are interested in the first 35,000 words of the corpora, in order to recreate the data science experiment of the hip-hop vocabulary; which determines the number of unique words used within an artist's first 35,000 lyrics.

```
>>len(the_corpus)
 112871
```

The corpus is stored in a list data structure, which lends itself to list manipulation techniques. You can see all the ways you can interact with a list in the Python documentation here https://docs.Python.org/2/tutorial/datastructures.html.

Did you know that 80-90% of time spent on data projects is gathering data and putting it into a format you can analyze? Geez

Let's take a look inside `the_corpus`, to determine what the first 10 words are.

```
>> the_corpus[:10]
['Dreamed',
'of',
'you',
'this',
'morning',
'Then',
'came',
'the',
'dawn',
'and']
```

Here's a little secret: much of NLP (and data science, for that matter) boils down to counting things. If you've got a bunch of data that needs analyzin' but you don't know where to start, counting things is usually a good place to begin. Sure, you'll need to figure out exactly what you want to count, how to count it, and what to do with the counts, but if you're lost and don't know what to do, just start counting. [1]

Lets slice the corpus down to the first 35,000 words.

[1] Some of this content has been adapted from Charlie Greenbacker's "A smattering of NLP in Python"

```
>> the_corpus[:35000]
```

We can now go ahead and figure out the number of unique words used in Jay Z's first 35,000 lyrics. An astute observer will notice that we have not done any data cleaning. For example, take a look inside a slice of the corpus, the last 10 words `the_corpus[34990:35000]`, `['calm', 'your', 'boys', 'Cause', 'I', 'm', 'findin', 'it', 'a', 'little']`, you will see it has treated the contraction "I'm" as two separate words. The `create_corpus` function that we used, works by separating each contiguous chunk of alphabets separated by punctuations or space as a word. As a result contractions like "I'm" gets treated as two words. We can use the function `lexical_diversity` to determine the number of unique words in our Jay Z corpus.

```
def lexical_diversity(my_text_data):
        word_count = len(my_text_data)
        vocab_size = len(set(my_text_data))
        diversity_score = word_count / vocab_size
        return diversity_score
```

If we call our function on the Jay Z sliced corpus, it should give us a score.

```
>>lexical_diversity(the_corpus[:35000])
 6
```

The lexical diversity of Jay Z's first 35,000 lyrics is 6

Exercise

1. Have your own fun, investigate who is more apt with the word, Jane Austen or Jay Z? You can find the code we have written so far in Lexical_Diversity.py file. Write your own code, to figure out the lexical diversity of the King James Bible and Jane Austen's Emma, as compared to Jay Z's.

2. Write a function named `NumberOfUniqueWords(SomeCorpus)` that determines the # of unique words used in a corpus. Test it on the King James Bible, Jane Austen's Emma and Jay Z's corpus. The function `NumberOfUniqueWords(SomeCorpus)` should take one argument (a corpus) and returns the computed result. Below are a few example inputs.

   ```
   >>> NumberOfUniqueWords(emma[:35000])
   3449
   >>> NumberOfUniqueWords(the_corpus[:35000])
   1036
   ```

A.5 Text Analysis for Meaningful Insights

Remember we had created a `wordlist` of type `PlaintextCorpusReader`. This uses the data structure of a nested list. PlaintextCorpusReader type is made up of a list of fileids, which are made up of a list of paragraphs, which are in turn made up of a list of sentences, which are in turn made up of a list of words.
`words()`: list of str
`sents()`: list of (list of str)
`paras()`: list of (list of (list of str))
`fileids()`: list of (list of (list of (list of str)))

```
>> Albums = wordlist.fileids()
>> Albums[:14]
['JayZ_American Gangster_American Dreamin.txt',
 'JayZ_American Gangster_American Gangster.txt',
 'JayZ_American Gangster_Blue Magic.txt',
 'JayZ_American Gangster_Fallin.txt',
 'JayZ_American Gangster_Hello Brooklyn 20.txt',
 'JayZ_American Gangster_I Know.txt',
 'JayZ_American Gangster_Ignorant Shit.txt',
 'JayZ_American Gangster_Intro.txt',
 'JayZ_American Gangster_No Hook.txt',
 'JayZ_American Gangster_Party Life.txt',
 'JayZ_American Gangster_Pray.txt',
 'JayZ_American Gangster_Say Hello.txt',
 'JayZ_American Gangster_Success.txt',
 'JayZ_American Gangster_Sweet.txt']
```

In this section, lets investigate the use of basketball language in Jay Z's lyrics. Go ahead and save all the album titles by typing in `Albums = wordlist.fileids()`. Notice that JayZ_ text appears before each of the filenames. To get this prefix out of the filename, we can extract the first five characters, using `fileid[5:]` .

```
[fileid[5:] for fileid in Albums[:14]]
```

We are ready to start mining the data. So let's do a simple analysis on the occurrence of the concept "basketball" in JayZ's lyrics as represented by a list of 40 terms that are common when we talk about basketball.

```
basketball_bag_of_words = ['bounce','crossover','technical',
 'shooting','double','jump','goal','backdoor','chest','ball',
 'team','block','throw','offensive','point','airball','pick',
```

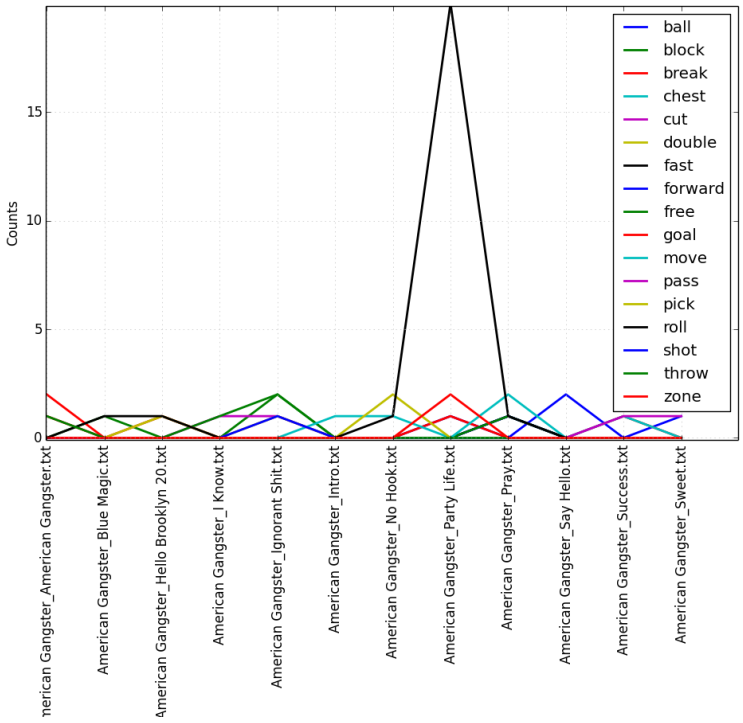
```
'assist','shot','layup','break','dribble','roll','cut','forward',
'move','zone','three-pointer','free','post','fast','blocking','backcourt',
'violation','foul','field','pass','turnover','alley-oop','guard']
```

Lets reduce our investigation of this concept to just the American Gangster album, which is the first 14 songs in the corpus. We do that by using the command `Albums[:14]`. Remember that `Albums` is just a list data type, so we can slice it, to its first 14 indexes.

The following code converts the words in the basket ball concept to lowercase using `w.lower()`, then checks if they start with any of the "targets", that is, each of the words in the `basketball_bag_of_words`, using the command `startswith()`. Thus, it will count words like "turnover," "alley-oop" and so on. All this is enabled by NLTK's built in function for Conditional Frequency Distribution. You can read more about it [here].

```
>> cfd = nltk.ConditionalFreqDist(
        (target, fileid[5:])
        for fileid in Albums[:14]
        for w in wordlist.words(fileid)
        for target in basketball_bag_of_words
        if w.lower().startswith(target))

>> cfd.plot()
```



Reflection

From the plot we see that the basketball term "roll" seems to be used extensively in the song *Party Life*. Let's take a closer look at this phenomenon, and determine if "roll" was used in the "basketball" sense of the term. To do this, we need to see the context in which it was used. What we really need is a concordance. Let's build one.

The first thing I want to do is to create a corpus that only contain words from the American Gangster album.

```
>> AmericanGangster_wordlist =
        PlaintextCorpusReader(corpus_root, 'JayZ_American Gangster_.*')
>> AmericanGangster_corpus =
        create_corpus(AmericanGangster_wordlist, [])
```

Concordance

Building a concordance, gets us to the area of elementary information retrieval (IR), think, *basic search engine*.

The first step in building our concordance is to do some data transformation, in this case, we need to *normalize* our terms. Why do we need to "normalize" our terms? We want to make sure that if we hit a term like "U.S.A." and "USA", that they correspond to the same concept. Further, when we enter "roll", we would like to *conceptually* match "roll", and "rolling".

One way to do this is to stem words. That is, reduce a word down to its base/stem/root form. As such "automate(s)", "automatic", "automation" all reduced to "automat". Most stemmers are pretty basic and just chop off standard affixes indicating things like tense (e.g., "-ed") and possessive forms (e.g., "-'s"). Here, we'll use the most popular English language stemmer, the Potter stemmer, which comes built-in with NLTK.

Once our words/tokens are stemmed, we can rest easy knowing that "roll", "Rolling", and "Rolls" will all stem to "roll".

```
>>porter = nltk.PorterStemmer()
>>stemmed_tokens = [porter.stem(t) for t in AmericanGangster_corpus]
>>for token in sorted(set(stemmed_tokens))[860:870]:
        print token + ' [' + str(stemmed_tokens.count(token)) + ']'
dummi [1]
dump [1]
dure [1]
each [1]
earlob [1]
eas [1]
```

```
easel [1]
easi [1]
easili [1]
eat [5]
```

Now we can go ahead and create a concordance to test if "roll" is used in the basketball (pick and roll) sense or not.

```
>>AmericanGangster_lyrics = IndexedText(porter, AmericanGangster_corpus)
>>AmericanGangster_lyrics.concordance('roll')
in my veins like a Pisces The Pyrex pot rolled up my sleeves Turn one into two l
dinner s now turn in to Breakfast I only roll Lexus to Hug your road I love your
ut I do lift Weight like I m using roids Rolls Royce Keep my movements smooth whi
lefty Gangster effortlessly Poppa was a  rolling stone its in my ancestry I m in
babe I got a slick mouth you might wanna roll with me I m on her bra strap she
ike CASINO They should pay me for some B roll Takin G strolls through the ghee to
n lets keep it smooth This that shit you roll up like a little tight J to Sip ya
her s afro as momma taps her toes as she rolls her jays and my poppa just left th
```

Based on the context, you can decide if the word "roll" is used in a basketball sense. This is really where the "art" of the word "Arts and Sciences" comes to play in Data Science and NLP[2].

[2]Some of this content has been adapted from Dan Jurafsky's Stanford CS124 class

Appendix B

Technical Implementation of Data Analysis

B.1 Survey Data

The analysis for this research followed the data science workflow that was discussed in Chapter 5.2.

Acquire data

The analysis was done using the Python programming language. The surveys were implemented using UC Berkeley's secure Google Apps' Google Form, which collected responses in a spreadsheet. The responses were downloaded as comma separated (CSV) files. The Pandas Python package was used to convert the csv files into a data frame from which analysis could be run. Two data frames were created, one for CS10, and the other for CS61A.

Reformat and clean data

The following series of actions were done on the data to get it ready for analysis.

- To facilitate analysis, all column names were renamed to match a priori names that I had created for the codebook of the data.
- The data for student's who did not consent to participate in the surveys were dropped from the table.
- Each data frame was sub-divided based on gender.
- Dimensions were created to focus analysis on the general attitudes that the survey was designed to measure.

```
atcs = ['atcs_1', 'atcs_2', 'atcs_3', 'atcs_4',
        'atcs_5', 'atcs_6', 'atcs_7', 'atcs_8',
        'atcs_9']# Attitude about CS competency

atcsgender = ['atcsgender_1', 'atcsgender_2', 'atcsgender_3']
atct = ['atct_1', 'atct_2', 'atct_3', 'atct_4', 'atct_5',
                'atct_6', 'atct_7', 'atct_8'] # Attitudes about CT

blg = ['blg_1', 'blg_2', 'blg_3',
        'blg_4'] # Sense of belonging in the class room
clet = ['clet_1', 'clet_2'] # Social implications and ethics
cltrcmp = ['cltrcmp_1', 'cltrcmp_2'] # Cultural competency
mtr = ['mtr_1', 'mtr_2', 'mtr_3'] # CS Mentors
prcs = ['prcs_1', 'prcs_2', 'prcs_3', 'prcs_4', 'prcs_5'] # Prior CS Exposure
gender = 'gender'
major = 'major'
priorcs10 = 'priorcs10' # Had taken CS10 prior
```

- In order to be able to make sense of the ordinal data, represented by a numeral in the range [1,5], I had to normalize the data, by projecting it onto a larger dimension of [1,100]; which allowed me to think of the responses as percentages.

```
def scaleData(theDataFrame):
    # I used the sum of everything so that we are comparing real percentages.
    theMin, theMax = theDataFrame.min(), theDataFrame.sum(axis=1)

    for (index, val) in theDataFrame.iteritems():
        theDataFrame[index] = ( val / theMax) * 100
    # Rounding errors don't make the percentages add up to 100,
    # sometimes its 98, 97 and so on
```

Create analysis scripts

Analysis scripts were created to test for:

- Statistical significance between students in CS10 and CS61a.
  - Student responses were plotted as bar charts along the following dimensions: blg, clet, cltrcmp, atcs, atcsgender, and atct.

- Statistical significance between students with prior exposure and students without prior exposure to CS in both classes
  - Similarly, student responses were plotted as bar charts along the following dimensions: blg, clet, cltrcmp, atcs, atcsgender, and atct.

B.2 Interview Data

Read in the data.

```
from pandas import DataFrame
# (*) Pandas for data manipulation
import pandas as pd

df = pd.read_csv('InterviewDataResponses.csv')
```

Rename columns for ease of understanding.

```
df.columns =  ['timestamp','howLearnComp', 'negAspectComp',
'dataLab','magicWand','csCourseFuture','currFactor',
'changeThoughtBerkeley','changePerceptnCS','funAspectClass',
'unfunAspectClass','respectCourseStaff','courseCulture61A',
'changeThoughtBerkeley2','moreCSClasses','perceptStudyCSChange',
'feelMinorityCS','whatFirstBroughtCourseAttention', 'race',
'reservationTakingClass','probStudyCS', 'comptThink',
'relationshipProgrammerScientist','enjoyCourse','enjoyProbSolv',
'priorCSbeforeClass','academicStrenghts','otherAcademicInterest',
'majorBerkeley','answerDiff','strengthCS','coolestAspectCS',
'thinkSomeOneCSDo','benefitStudyCS','theName','priorExposureToCS',
'cs10','cs61A', 'gender', 'acadClass','experienceSnap',
'anythingLikeToShare','extra','expectGetOutOfCourse',
'q_CT','q2','q3','q4','q5','q6','q7','q8','q9','q10',
'q11','q12','q13','q14','q15','q16','q17']
```

Break up the data into several data frames representing CS10, CS61A, and then each by gender.

```
cs10 = df[df.cs10 == 'yes']
cs61a = df[df.cs61A == 'yes']
cs10.reset_index(drop=True)
cs61a.reset_index(drop=True)

cs10_female = cs10[cs10.gender == 'female' ]
```

```
cs10_male = cs10[cs10.gender == 'male']
cs10_female = cs10_female.reset_index(drop=True)
cs10_male = cs10_male.reset_index(drop=True)


cs61a_female = cs61a[cs61a.gender == 'female']
cs61a_female = cs61a_female[pd.isnull(cs61a_female.cs10)]

cs61a_male = cs61a[cs61a.gender == 'male']
cs61a_male = cs61a_male[pd.isnull(cs61a_male.cs10)]
cs61a_female = cs61a_female.reset_index(drop=True)
cs61a_male = cs61a_male.reset_index(drop=True)
```

The follow section contains functions developed from NLTK used to build the key phrase extractor, that I deployed to code the data. The information I used was guided by the work of Alex Bowe, http://alexbowe.com/au-naturale/.

```
import nltk
from nltk.corpus import stopwords
stopwords = stopwords.words('english')


def leaves(tree):
    """Finds NP (nounphrase) leaf nodes of a chunk tree."""
    for subtree in tree.subtrees(filter = lambda t: t.node=='NP'):
        yield subtree.leaves()

def normalise(word):
    """Normalises words to lowercase and stems and lemmatizes it."""
    word = word.lower()
    #word = stemmer.stem_word(word)
    word = lemmatizer.lemmatize(word)
    return word

def acceptable_word(word):
    """Checks conditions for acceptable word: length, stopword."""
    accepted = bool(2 <= len(word) <= 40
        and word.lower() not in stopwords)
    return accepted
```

```
def get_terms(tree):
    for leaf in leaves(tree):
        term = [ normalise(w) for w,t in leaf if acceptable_word(w) ]
        yield term

# Used when tokenizing words
sentence_re = r'''(?x)      # set flag to allow verbose regexps
      ([A-Z])(\.[A-Z])+\.?  # abbreviations, e.g. U.S.A.
    | \w+(-\w+)*            # words with optional internal hyphens
    | \$?\d+(\.\d+)?%?      # currency and percentages, e.g. $12.40, 82%
    | \.\.\.                # ellipsis
    | [][.,;"'?():-_`]      # these are separate tokens
'''

lemmatizer = nltk.WordNetLemmatizer()
stemmer = nltk.stem.porter.PorterStemmer()


# Grammer taken from S. N. Kim, T. Baldwin, and M-Y. Kan.
# Evaluating n-gram based evaluation metrics for automatic keyphrase extraction.

grammar = r"""
    NBAR:
        {<NN.*|JJ>*<NN.*>}  # Nouns and Adjectives, terminated with Nouns

    NP:
        {<NBAR>}
        {<NBAR><IN><NBAR>}  # Above, connected with in/of/etc...
    """

 def iterateCodes(dfName, category, item):


        dfCol = []
        dfCol.append(dfName+category+'.columns')

        code_tmp = {}

        dfItem = []
        dfItem.append(dfName+category+'.iterrows()')
        for row, columns in eval(dfItem[0]):
            if pd.isnull(columns[item]):
```

```
                continue
            else:
                text = str(columns[item]).split()
                text = ' '.join(text)
                chunker = nltk.RegexpParser(grammar)
                toks = nltk.regexp_tokenize(text, sentence_re)
                postoks = nltk.tag.pos_tag(toks)
                tree = chunker.parse(postoks)
                terms = get_terms(tree)

                code = []
                for term in terms:
                    for word in term:
                        if code_tmp.has_key(word):
                            code_tmp[word] += 1
                        else:
                            code_tmp[word] = 0



        return code_tmp


def printCode(coding):

    window = 10

    for key in sorted(coding):
        if coding[key] > 1 and (window > 0 and window <=10):
            print "%s" % (key),
            window = window - 1
        elif window < 1:
            print '\n'
            window = 10
    print '\n------- E N D--------\n\n'
```

Now I can iteratively go through the data and see what the key phrases are for each chunk of text.

```
coding = {}
item = 'feelMinorityCS'
coding = iterateCodes('cs10', '', item)
```

```
print "CS10\n"
printCode(coding)

coding = iterateCodes('cs61a', '', item)
print"CS61A\n"
printCode(coding)
```

```
CS10

asian bit c class computer course cs10 different doesn everyone

experience girl guy lot mac minority partner people project re

science ta thing time way woman yes
------- E N D--------


CS61A

asian berkeley bit c class community computer course cs10 different

doesn eec everybody everyone experience feel first girl guy kind

lab lot major minority oh partner people person pm project

re science sense student stuff thing time way white woman

yeah yes
------- E N D--------
```

Appendix C

Informed Consent Form

Consent to Participate in Research
HipHopathy - A HipHop Data Science Module on Students Perception and Persistence in Computer Science

Introduction and Purpose My name is Omoju Miller. I am a graduate student at the University of California, Berkeley working with my faculty advisor, Professor Alice Agogino in the School of Science and Mathematics Education (Computer Science Education). I would like to invite you to take part in my research study, which concerns investigating social and curricular factors that lead to student retention and attrition in introductory Computer Science at UC Berkeley.

Procedures If you agree to participate in my research, I will conduct an interview with you at a time and location of your choice. The interview will involve questions about your experience with regards to Computer Science. It should last about 30 minutes. With your permission, I will audiotape and take notes during the interview. The recording is to accurately record the information you provide, and will be used for transcription purposes only. If you choose not to be audio-taped, I will take notes instead. If you agree to being audiotaped but feel uncomfortable at any time during the interview, I can turn off the recorder at your request. Or if you don't wish to continue, you can stop the interview at any time. I expect

to conduct only one interview; however, follow-ups may be needed for added clarification. If so, I will contact you by email/phone to request this.

Risks/Discomforts Some of the research questions may make you uncomfortable or upset. You are free to decline to answer any questions you don't wish to, or to stop the interview at any time. I don't believe that interviews will trigger such a response, because questions will be mainly concerned with your opinions and preferences. As with all research, there is a chance that confidentiality could be compromised; however, I am taking precautions to minimize this risk.

Measures to Minimize risk I will comply with all UC Berkeley requirements for computer security including the CPHS Data Security Policy. The data collected will not include your social security numbers or other financially sensitive information.

Plans for reporting unintended problems An initial report will be made by fax, mail/delivery, phone, email, to the Director, Research Subject Protection as soon as possible, but within no more than one week (7 calendar days) of the Principal Investigator learning of the incident. The initial report will be followed by a formal written report within no more than two weeks (14 calendar days) of the Principal Investigator learning of the incident.

Benefits There is no direct benefit to you from taking part in this study. It is hoped that the research will help more students get into the Computer Science field. Further students will benefit from the study because it will make them better prepared to succeed in their future CS courses. Society will benefit from the study, as it is designed to help stem attrition, which may result in increased Computer Science graduates.

Confidentiality Your study data will be handled as confidentially as possible. If results of this study are published or presented, individual names and other personally identifiable information will not be used. To minimize the risks to confidentiality, electronic data with personal identifiers will be encrypted and stored on a UC Berkeley approved box storage cloud. Furthermore, research files stored on box will be encrypted, and password protected. Much of this electronic data will be gathered from EDx servers, hosted at UC Berkeley. There are a variety of ways that the data will be transferred from EDx to us. In all cases, only subjects who have assented/consented will have data transferred. And, once data is transferred, it will be stored per the description above.

When the research is completed, I may save the tapes and notes for use in future research done by myself or others. I will retain these records for up to 3 years after the study is over. The same measures described above will be taken to protect confidentiality of this study data.

Compensation You will not be paid for taking part in this study.

Rights Participation in this research is completely voluntary. You are free to decline to take part in the project. You can decline to answer any questions and are free to stop taking part in the project at any time. Whether or not you choose to participate in the research and whether or not you choose to answer a question or continue participating in the project, there will be no penalty to you or loss of benefits to which you are otherwise entitled.

Questions If you have any questions about this research, please feel free to contact me. I can be reached at (650) 396-9743 or omojumiller@berkeley.edu.
If you have any questions about your rights or treatment as a research participant in this study, please contact the University of California at Berkeley's Committee for Protection of Human Subjects at 510-642-7461, or e-mail subjects@berkeley.edu.

CONSENT If you agree to participate, please do so by clicking the continue button on this survey. You will be given a copy of this form to keep for your own records.

Appendix D

Interview Protocol

These interview protocol questions were adapted from the Master Thesis of Lily Irani, Stanford University (Irani, 2002, pp. 68). While these questions are meant as a guide for conversation to ensure certain topics are covered in the interview, the interviewer is free to add investigatory questions as issues arise during the interview, or eliminate questions that will result in redundancy.

Student background
This questions will be asked once, the first time the interviewer meets with a student.

- Can you tell me the story about you and computers? (How and when did you get interested? How do you typically use them?)
  Purpose: Get a sense of specific experiences and exposure that shaped their interest in computers, if they are interested. If they dont have a strong interest, learn what experience of lack thereof contributed to that.
- In your opinion, what are positive and negative aspects of computers?
  Purpose: Probe student's attitude towards computers.
- How does your family use computers?
  Purpose: Was there a computer expert in the family? Who was it? How did parents use it? What is parents' occupation? Brothers and sisters attitude towards computing?

Academic interests
These questions will be asked once, the first time the interviewer meets with a student.

- What do you consider to be your academic strengths? Other interests?
- What are you considering majoring in at Berkeley?
- What about minors or other academic interests?

- Would your answer to that question have been different if Id asked when you arrived on campus?

Perceptions of Computer Science
These questions will be asked once, the first time the interviewer meets with a student.

- What strengths do you think a successful computer science major should have?
- In your estimation, what is the coolest aspect of CS?
- What do you think someone can do with a degree in CS?
- What do you perceive as benefits to studying CS?
- What do you perceive as problems in studying CS?
- What is the relationship between a Computer Scientist and a Programmer?
- Lets talk about Computational thinking?
  Do you know what it is?
- Enjoyment of the course?
- Enjoyment of problem solving?
- Emotional response somehow what did it leave them feeling like
- Do you ever feel like a minority in any way in the CS department?
- Ask what their status is, freshmen, sophomore and so on

For CS10 Students

- What first brought this course to your attention?
  Purpose: what factors in making this decision, for example: Academic advising? Prior interest? Prior courses in the sequence (if applicable)?
- Why are you taking this course? Any other reasons?
  Purpose: Probe for place of course and material in terms of students goals? Perceptions of course? Course reputation?
- Did you have any reservations about taking this class?
  Purpose: What influences the students decision to take the course? How does reputation affect girls decision to take the class?

- What do you expect get out of this course?
  Purpose: What do students find to be attractive objectives in a course? practical knowledge? conceptual understanding? literacy? idea of CS as a field?
- What CS courses do you plan to take?
  Purpose: Gives sense of a change over time if these plans change.
- Were there any curricular factors in the course that stood out to you?
  Purpose: Determine if there are curricular factors that impact students attitude towards CS
- Did taking this course change your thoughts on what you might want to take while here at Berkeley? How? Why?
  Listen for: modified perceptions of CS and CS majors? what altered interest? change in intent to major or minor?
- Did you find that your perceptions of what it is to study CS changed that semester?
  Purpose: See if they realize that computational thinking is what they are learning, and not a programming language.
- Did you find that your perceptions of what it is to be a Computer Scientist changed that semester?

For CS61A Students who had previously taken CS10

- Did you enjoy taking [CS10, CS61A]?
- What aspects of your experience did you find particularly fun or rewarding?
  Anything else?
- What aspects of your experience did you find unfavorable or frustrating?
  Anything else?
- Did you feel respected by the instructor? the course staff? your peers?
  What gives you those impressions?
  Purpose: are there subtly (or unsubtly) biased comments coloring girls experiences?
- How would you define course culture of CS 61A?
  Did you feel as if you fit in?
  Purpose: how does culture change from course to course? how does this affect girls sense of belonging?
- Were there any curricular factors in the course that stood out to you?
  Purpose: Determine if there are curricular factors that impact students attitude towards CS

- Did taking this course change your thoughts on what you might want to take while here at Berkeley? How? Why?
  Listen for: modified perceptions of CS and CS majors? what altered interest? change in intent to major or minor?
- Do you think you will take any other CS courses? Why or not? What course?
- Did you find that your perceptions of what it is to study CS changed that semester?
  Purpose: See if they realize that computational thinking is what they are learning, and not a programming language.
- Did you find that your perceptions of what it is to be a Computer Scientist changed that semester?

Appendix E

Survey Instruments

Demographics

- Year in university [Freshman, Sophomore, Junior, Senior]
- Gender [Male, Female, Other]
- What is your reason for taking this class [interested, other]
- What is your major?

Attitudes towards Computer Science

- I like to use Computer Science to solve problems.
- Knowledge of computing will allow me to secure a good job.
- I can learn to understand computing concepts.
- My career goals do not require that I learn computing skills.
- I can achieve good grades (C or better) in computing courses.
- I do not like using computer science to solve problems.
- I am confident that I can solve problems by using computer applications.
- The challenge of solving problems using computer science appeals to me.
- I am comfortable with learning computing concepts.
- I would take additional Computer Science courses if I were given the opportunity.
- I am confident about my abilities with regards to computer science.
- I do think I can learn to understand computing concepts.

Attitudes about Computational Thinking

- I am good at solving a problem by thinking about similar problems I've solved before.
- I have good research skills.
- I am good at using online search tools.
- I am persistent at solving puzzles or logic problems.
- I know how to write computer programs.
- I am good at building things.
- I'm good at ignoring irrelevant details to solve a problem.
- I know how to write a computer program to solve a problem.
- I work well in teams.
- I think about the ethical, legal, and social implications of computing.

Computer Science Mentors and Role Models

- Before I came to UC Berkeley, I knew people who have careers in Computer Science.
- There are people with careers in Computer Science who look like me.
- I have role models within the Computer Science field that look like me.

Identity and Self Efficacy

- In this class, I feel I belong.
- In this class, I feel awkward and out of place
- In this class, I feel like my ideas count
- In this class, I feel like I matter.
- I am comfortable interacting with peers from different backgrounds than my own (based on race, sexuality, etc.)
- I have good cultural competence, or the ability to interact effectively with people from diverse backgrounds.
- Our class materials (e.g., case studies and projects) were relevant and practical

Gendered Belief about Computer Science Ability

- Women are less capable of success in CS than men
- Men have better math and science abilities than women.
- Women are smarter than men.

Pre-Collegiate CS Preparation

- Did you take a CS course in High School?
- Did you have exposure to Computer Science before UC Berkeley?
- Did a family member introduce you to Computer Science?
- Did you have a close family member who is a Computer Scientist or is affiliated with computing industry?
- Did your school offer AP CS?
- How prepared did you feel about this class before it started?
- Will you be taking any more CS classes (if so which ones?)
- (For 61A only) Have you taken CS10, The Beauty and Joy of Computing?

Bibliography

[ACM12] Association of Computing Machinery. *K-12 Computer Science Education: Unlocking the Future of Students*. Tech. rep. ACM, 2012. URL: http://www.acm.org/public-policy/education-policy-committee/2012_CS_Slides_Aug.pptx (cit. on pp. 3, 6).

[Aho11] A. V. Aho. "What is Computation?" In: *Ubiquity* (2011), pp. 1–8 (cit. on p. 19).

[Alt72] F. L. Alt. "Archaeology of Computers: Reminiscences, 1945-1947". In: *Commun. ACM* 15.7 (July 1972), pp. 693–694 (cit. on p. 2).

[Ast+09] O. Astrachan, H. Walker, C. Stephenson, L. Diaz, and J. Cuny. "Advanced Placement Computer Science: The Future of Tracking the First Year of Instruction". In: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. SIGCSE '09. Chattanooga, TN, USA: ACM, 2009, pp. 397–398 (cit. on p. 2).

[Ast+11a] O. Astrachan, T. Barnes, D. D. Garcia, J. Paul, B. Simon, and L. Snyder. "CS Principles: Piloting a New Course at National Scale". In: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*. SIGCSE '11. Dallas, TX, USA: ACM, 2011, pp. 397–398 (cit. on pp. 8, 57, 58).

[Ast+11b] O. Astrachan, J. Cuny, C. Stephenson, and C. Wilson. "The CS10K Project: Mobilizing the Community to Transform High School Computing". In: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*. SIGCSE '11. Dallas, TX, USA: ACM, 2011, pp. 85–86 (cit. on p. 51).

[Atw06] J. Atwood. *Separating Programming Sheep from Non-Programming Goats*. 2006. URL: http://blog.codinghorror.com/separating-programming-sheep-from-non-programming-goats/ (cit. on p. 38).

[Atw15] J. Atwood. *Jeff Atwood on building Discourse, Stack Exchange, and Coding Horror*. 2015. URL: https://scaleyourcode.com/interviews/interview/10 (cit. on p. 66).

[BD06] R. Bornat and S. Dehnadi. *The Camel has Two Humps*. Working Paper. Middlesex University, 2006. URL: http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf (cit. on p. 38).

[Ber14] N. Berg. "Predicting crime, LAPD-style". In: *The Guardian* (2014). URL: http://www.theguardian.com/cities/2014/jun/25/predicting-crime-lapd-los-angeles-police-data-analysis-algorithm-minority-report (cit. on p. 66).

[BLS12] Bureau of Labor Statistics (BLS). *Employment Projections 2010-2020*. 2012. URL: http://www.bls.gov/emp/ (cit. on p. 6).

[Blu+07] L. Blum, C. Frieze, O. Hazzan, and M. B. Dias. "A Cultural Perspective on Gender Diversity in Computing". In: *Reconfiguring the Firewall: Recruiting Women to Information Technology across Cultures and Continents*. Ed. by C. J. Burger, E. G. Creamer, and P. S. Meszaros. Wellesley, MA: A K Peters CRC Press, 2007, pp. 109–134 (cit. on p. 36).

[Bor+08] R. Bornat, S. Dehnadi, and Simon. "Mental Models, Consistency and Programming Aptitude". In: *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*. ACE '08. Wollongong, NSW, Australia: Australian Computer Society, Inc., 2008, pp. 53–61 (cit. on p. 38).

[BR12] K. Brennan and M. Resnick. "New frameworks for studying and assessing the development of computational thinking". In: *American Educational Research Association*. 2012 (cit. on pp. 19, 21).

[Bro14] K. V. Brown. "Tech shift: More women in computer science classes". In: *San Francisco Chronicle* (2014). URL: http://www.sfgate.com/education/article/Revamped-computer-science-classes-attracting-more-5243026.phpt (cit. on p. 59).

[BS11] V. Barr and C. Stephenson. "Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?" In: *ACM Inroads* 2.1 (Feb. 2011), pp. 48–54 (cit. on p. 19).

[Bus+13] K. Buse, D. Bilimoria, and S. Perelli. "Why They Stay: Women Persisting in US Engineering Careers". In: *Career Development International* 18.2 (2013), pp. 139–154 (cit. on p. 131).

[CA00] J. Cuny and W. Aspray. *Recruitment and retention of women graduate students in computer science and engineering*. Tech. rep. 2. Washington, DC: The Computing Research Association, June 2000 (cit. on p. 32).

[Cam97] T. Camp. "The Incredible Shrinking Pipeline". In: *Commun. ACM* 40.10 (Oct. 1997), pp. 103–110 (cit. on p. 32).

[Cas15] B. Casselman. "Where Police Have Killed Americans In 2015". In: *FiveThirtyEight* (2015). URL: http://fivethirtyeight.com/features/where-police-have-killed-americans-in-2015/ (cit. on p. 71).

[Che+09] S. Cheryan, V. C. Plaut, P. G. Davies, and C. M. Steele. "Ambient belonging: how stereotypical cues impact gender participation in computer science." In: *Journal of personality and social psychology* 97.6 (Dec. 2009), pp. 1045–60 (cit. on pp. 25, 131, 132).

[Cod15] Code.org. *Hour of Code*. 2015. URL: https://hourofcode.com/us (cit. on p. 36).

[Cre03] J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 2nd. Vol. 3. Sage publications, 2003. Chap. 1 (cit. on p. 42).

[Cro05] S. Crossley. "Metaphorical Conceptions in Hip-Hop Music". In: *African American Review* 39.4 (2005), pp. 501–512 (cit. on p. 74).

[Das12] S. Dasgupta. "Pedagogical Inspirations". In: *Learning with Data; A toolkit to democratize the computational exploration of data*. 2012. Chap. 3 (cit. on p. 75).

[Den09] P. J. Denning. "The profession of IT Beyond computational thinking". In: *Communications of the ACM* 52.6 (June 2009), p. 28 (cit. on pp. 19, 20).

[DeN15] J. DeNero. *Foundations of Data Science Intro*. 2015. URL: http://databears.berkeley.edu/ (cit. on p. 68).

[Dew15] C. Dewey. "Google Maps' White House glitch, Flickr auto-tag, and the case of the racist algorithm". In: *The Washington Post* (2015). URL: https://www.washingtonpost.com/news/the-intersect/wp/2015/05/20/google-maps-white-house-glitch-flickr-auto-tag-and-the-case-of-the-racist-algorithm/ (cit. on p. 64).

[DiS12] E. DiSalvo. "Glitch Game Testers: The design and study of a learning environment for computational production with young African American males". PhD thesis. Georgia Institute of Technology, 2012 (cit. on p. 62).

[DL88] C. S. Dweck and E. L. Leggett. "A social-cognitive approach to motivation and personality." In: *Psychological Review* 95.2 (1988), pp. 256–273 (cit. on p. 39).

[Dol90] J. Dollard. "The Dozens: Dialectic of Insult". In: *Mother Wit from the Laughing Barrel*. Second. Jackson, MS: University Press of Mississippi, 1990, 277–6=294 (cit. on p. 31).

[Dri+00] R. Driver, P. Newton, and J. Osborne. "Establishing the Norms of Scientific Argumentation in Classrooms". In: *Science Education* 84.3 (2000), pp. 287–312 (cit. on p. 70).

[Egl+06] R. Eglash, A. Bennett, C. O'Donnell, S. Jennings, and M. Cintorino. "Culturally Situated Design Tools: Ethnocomputing from Field Site to Classroom". In: *American Anthropologist* 108.2 (2006), pp. 347–362 (cit. on pp. 13, 27).

[Egl+11] R. Eglash, M. Krishnamoorthy, J. Sanchez, and A. Woodbridge. "Fractal Simulations of African Design in Pre-College Computing Education". In: *Trans. Comput. Educ.* 11.3 (Oct. 2011), 17:1–17:14 (cit. on pp. 27, 149).

[Egl+13] R. Eglash, J. E. Gilbert, and E. Foster. "Toward culturally responsive computing education". In: *Commun. ACM* 56.7 (July 2013), pp. 33–36 (cit. on pp. 13, 27, 30).

[Egl07] R. Eglash. "The Fractals at the Heart of African Designs". In: *TED* (2007). URL: http://www.ted.com/talks/ron_eglash_on_african_fractals (cit. on pp. 27, 28).

[Egl99] R. Eglash. "Fractals in African Settlement Architecture". In: *African Fractals: Modern Computing and Indigenous Design*. Rutgers University Press, 1999. Chap. 2 (cit. on pp. 29, 30).

[Eri+07] K. A. Ericsson, R. W. Roring, and K. Nandagopal. "Giftedness and evidence for reproducibly superior performance: an account based on the expert performance framework". In: *High Ability Studies* 18.1 (June 2007), pp. 3–56 (cit. on p. 37).

[Fer14] G. Ferenstein. "Women Outnumber Men For The First Time In Berkeley's Intro To Computer Science Course". In: *TechCrunch* (2014). URL: http://techcrunch.com/2014/02/21/women-outnumber-men-for-the-first-time-in-berkeleys-intro-to-computer-science-course (cit. on p. 59).

[Fis+97] A. Fisher, J. Margolis, and F. Miller. "Undergraduate women in Computer Science: experience, motivation and culture". In: *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education (SIGCSE '97)*. Ed. by J. E. Miller. New York, NY: ACM Press, 1997, pp. 106–110 (cit. on pp. 33, 132).

[Fol04] D. Foley. "Ogbu's theory of academic disengagement: its evolution and its critics". In: *Intercultural Education* 15.4 (2004), pp. 385–397 (cit. on p. 24).

[Fre00] P. Freire. *Pedagogy of the Oppressed*. 30th Anniversary. New York, NY 10010: Continuum International Publishing Group, 2000 (cit. on p. 147).

[Gar+11] D. Garcia, B. Harvey, and T. Barnes. *FRABJOUS CS - Framing a Rigorous Approach to Beauty and Joy for Outreach to Underrepresented Students in Computing at Scale*. Tech. rep. Berkeley, CA: UC Berkeley, 2011 (cit. on p. 52).

[Gar+12] D. D. Garcia, B. Harvey, and L. Segars. "CS Principles Pilot at University of California, Berkeley". In: *ACM Inroads* 3.2 (June 2012), pp. 58–60 (cit. on p. 51).

[Gar+14] D. D. Garcia, B. Harvey, T. Barnes, D. Armendariz, J. McKinsey, Z. MacHardy, O. Miller, B. Peddycord III, E. Lemon, S. Morris, and J. Paley. "AP CS Principles and the Beauty and Joy of Computing Curriculum (Abstract Only)". In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: ACM, 2014, pp. 746–746 (cit. on p. 51).

[GC02] D. Gürer and T. Camp. "An ACM-W literature review on women in computing". In: *ACM SIGCSE Bulletin* 34.2 (June 2002), pp. 121–127 (cit. on pp. 1, 32).

[Goo15] Google, Inc. *Google Diversity*. 2015. URL: www.google.com/diversity/index.html#chart (cit. on p. 6).

[GP13] S. Grover and R. Pea. "Computational Thinking in K-12: A Review of the State of the Field". In: *Educational Researcher* 42.1 (2013), pp. 38–43 (cit. on pp. 10, 20–22).

[Gre+96] J. G. Greeno, A. M. Collins, and L. B. Resnick. "Cognition and Learning". In: *Handbook of Educational Psychology*. Ed. by D Berliner and R Calfee. Vol. 1968. New York, NY: Macmillan, 1996. Chap. 2, pp. 15–46 (cit. on p. 11).

[Guo13] P. Guo. *Data Science Workflow: Overview and Challenges*. Online Blog of The Communications of the ACM. 2013 (cit. on p. 76).

[Har+14] B. Harvey, D. D. Garcia, T. Barnes, N. Titterton, O. Miller, D. Armendariz, J. McKinsey, Z. Machardy, E. Lemon, S. Morris, and J. Paley. "*Snap!* (Build Your Own Blocks) (Abstract Only)". In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: ACM, 2014, pp. 749–749 (cit. on p. 60).

[Har12] S. J. Harper. "The Transformation of Big Law Firms". In: *The Lawyer Bubble: A Profession in Crisis*. Basic Books, 2012. Chap. 5 (cit. on pp. 1, 7).

[Har14a] Q. Hardy. *Jesse Jackson Confronts Silicon Valley*. 2014. URL: http://bits.blogs.nytimes.com/2014/03/19/jesse-jackson-confronts-silicon-valley/ (cit. on p. 1).

[Har14b] B. Harvey. "Whatever Happened to the Revolution, Part 2: In Which I get Seduced By the Lure of a National Curriculum". In: *Constructionism and Creativity*. 2014 (cit. on pp. 19, 22).

[Hen14] S. Henn. "When Women Stopped Coding". In: *National Public Radio* (2014). URL: http://www.npr.org/sections/money/2014/10/21/357629765/when-women-stopped-coding (cit. on p. 37).

[HM10] B. Harvey and J. Mönig. "Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?" In: *Proceedings of Constructionism 2010*. Paris, France, 2010 (cit. on pp. 59, 60).

[Ira02] L. Irani. "A Different Voice: Women Exploring Stanford Computer Science". MA thesis. Stanford University, 2002 (cit. on p. 179).

[Isb+09] C. L. Isbell, L. A. Stein, R. Cutler, J. Forbes, L. Fraser, J. Impagliazzo, V. Proulx, S. Russ, R. Thomas, and Y. Xu. "(Re)defining computing curricula by (re)defining computing". In: *SIGCSE Bulletin* 41.4 (Jan. 2009), pp. 195–207 (cit. on p. 19).

[Jac14] J. Jackson. *Silicon Valley has a Proud Record on Innovation, A Shameful One on Equality*. 2014. URL: http://www.theguardian.com/commentisfree/2014/sep/21/silicon-valley-innovation-inequality-technology-jesse-jackson (cit. on p. 1).

[Kau06] J. C. Kaufman. "Self-reported differences in creativity by ethnicity and gender". In: *Applied Cognitive Psychology* 20.8 (2006), pp. 1065–1082 (cit. on pp. 30, 31).

[Kit+08] J. Kitzinger, J. Haran, M. Chimba, and T. Boyce. *Role Models in the Media: An Exploration of the Views and Experiences of Women in Science, Engineering and Technology*. Tech. rep. Cardiff School of Journalism, Media and Cultural Studies. Cardiff University, 2008 (cit. on p. 4).

[KM06] P. Kinnunen and L. Malmi. "Why Students Drop out CS1 Course?" In: *Proceedings of the Second International Workshop on Computing Education Research*. ICER '06. Canterbury, United Kingdom: ACM, 2006, pp. 97–108 (cit. on pp. 4, 25).

[Knu74] D. E. Knuth. "Computer Programming as an Art". In: *Communications of the ACM* 17.12 (1974), pp. 667–673 (cit. on p. 68).

[Kuh+00] D. Kuhn, R. Cheney, and M. Weinstock. "The Development of Epistemological Understanding". In: *Cognitive Development* 15.3 (2000), pp. 309 –328 (cit. on p. 70).

[LB95] G. Ladson-Billings. "But That's Just Good Teaching! The Case for Culturally Relevant Pedagogy". In: *Theory Into Practice* 34.3 (1995), pp. 159–165 (cit. on pp. 141, 147).

[Lec12] S. Leckart. "The Stanford Education Experiment Could Change Higher Learning Forever". In: *Wired* (2012). URL: http://www.wired.com/2012/03/ff_aiclass/ (cit. on p. 18).

[Lee08] C. D. Lee. "Synthesis of Research on the Role of Culture in Learning Among African American Youth: The Contributions of Asa G. Hilliard, III". In: *Review of Educational Research* 78.4 (2008), pp. 797–827 (cit. on p. 13).

[Lee93] C. D. Lee. *Signifying as a scaffold for literary interpretation: The pedagogical implications of an African American discourse genre*. Tech. rep. 26. Urbana, IL: National Council of Teachers of English, 1993, pp. 1–206 (cit. on p. 30).

[Lee95] C. D. Lee. "Signifying as a Scaffold for Literary Interpretation". In: *Journal of Black Psychology* 21.4 (1995), pp. 357–381 (cit. on pp. 13, 30).

[Lew+14] C. Lewis, S. Esper, V. Bhattacharyya, N. Fa-Kaji, N. Dominguez, and A. Schlesinger. "Children's Perceptions of What Counts As a Programming Language". In: *J. Comput. Sci. Coll.* 29.4 (Apr. 2014), pp. 123–133 (cit. on p. 62).

[Lew10a] C. Lewis. *Attrition in Introductory Computer Science at the University of California, Berkeley*. Tech. rep. UCB/EECS-2010-132. EECS Department, University of California, Berkeley, 2010. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-132.html (cit. on pp. 55, 58).

[Lew10b] C. M. Lewis. "How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch". In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. SIGCSE '10. Milwaukee, Wisconsin, USA: ACM, 2010, pp. 346–350 (cit. on p. 62).

[Lew14] C. Lewis. *Programming in Scratch*. 2014. URL: https://www.edx.org/course/programming-scratch-harveymuddx-cs002x-0#! (cit. on p. 52).

[Lin+10] M. Linn, A. V. Aho, M. B. Blake, R. Constable, Y. B. Kafai, J. L. Kolodner, L. Snyder, and U. Wilensky. *Report of a Workshop on The Scope and Nature of Computational Thinking*. Tech. rep. Washington, DC: National Research Council, 2010 (cit. on p. 19).

[Lis11] R. Lister. "Concrete and Other neo-Piagetian Forms of Reasoning in the Novice Programmer". In: *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*. ACE '11. Perth, Australia: Australian Computer Society, Inc., 2011, pp. 9–18 (cit. on p. 38).

[Mag+13] B. Magerko, J. Freeman, T. McKlin, S. McCoid, T. Jenkins, and E. Livingston. "Tackling engagement in computing with computational music remixing". In: *Proceeding of the 44th ACM technical symposium on Computer science education*. SIGCSE '13. Denver, Colorado, USA: ACM, 2013, pp. 657–662 (cit. on pp. 13, 30).

[Mar+00] J. Margolis, A. Fisher, and F. Miller. "The Anatomy of Interest : Women in Undergraduate Computer Science". In: *Womens Studies Quarterly* 28.1/2 (2000), pp. 104–127 (cit. on pp. 33, 37).

[Mar08] J. Margolis. *Stuck in the Shallow End: Education, Race, and Computing*. The MIT Press, 2008 (cit. on p. 23).

[Mat+11] K. Mattern, E. Shaw, and M. Ewing. *Advanced Placement Exam Participation: Is AP Exam Participation and Performance Related to Choice of College Major?* Tech. rep. College Board, 2011 (cit. on p. 2).

[Met04] K. E. Metz. "Children's Understanding of Scientific Inquiry: Their Conceptualization of Uncertainty in Investigations of Their Own Design". In: *Cognition and Instruction* 22.2 (2004), pp. 219–290 (cit. on p. 69).

[Mil14] O. Miller. "It's Deeper Than Rap, Toward Culturally Responsive CS". In: *XRDS* 20.4 (June 2014), pp. 28–30 (cit. on p. 68).

[Min70] M. Minsky. "Form and Content in Computer Science". In: *Journal of Association for Computing Machinery* 17.2 (1970), pp. 197–215 (cit. on p. 39).

[MM12] B. Moyers and K. Muhammad. "Khalil Muhammad on Facing Our Racial Past." In: *Moyers & Company* (2012). URL: http://billmoyers.com/segment/khalil-muhammad-on-facing-our-racial-past/ (cit. on p. 65).

[Mol+92] L. C. Moll, C. Amanti, D. Neff, and N. Gonzalez. "Funds of knowledge for teaching: Using a qualitative approach to connect homes and classrooms". In: *Theory Into Practice* 31.2 (1992), pp. 132–141 (cit. on p. 13).

[MS13] A. Martin and A. Scott. *Engaging Underrepresented Students in Computer Science: Examining the Effectiveness of a 5-week Computer Science Course in the SMASH Summer Academy*. Tech. rep. The Level Playing Field Institute, 2013 (cit. on p. 45).

[MW47] H. B. Mann and D. R. Whitney. "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other". In: *The Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60 (cit. on p. 46).

[NS76] A. Newell and H. A. Simon. "Computer Science as Empirical Inquiry: Symbols and Search". In: *Communications of the ACM* 19.3 (1976), pp. 113–126 (cit. on pp. 68, 73).

[NSF13] National Science Foundation. *Women, Minorities, and Persons with Disabilities in Science and Engineering: 2013*. Tech. rep. National Science Foundation, 2013 (cit. on p. 1).

[OS98] J. U. Ogbu and H. D. Simons. "Voluntary and Involuntary Minorities: A Cultural-Ecological Theory of School Performance with Some Implications for Education". In: *Anthropology & Education Quarterly* 29.2 (1998), pp. 155–188 (cit. on p. 23).

[Pap80a] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY: Basic Books, Inc, 1980 (cit. on pp. 11, 12).

[Pap80b] S. Papert. "Powerful Ideas in Mind-Sized Bites". In: *Mindstorms: Children, Computers and Powerful Ideas*. New York, NY: Basic Books, 1980, pp. 135–155 (cit. on p. 70).

[Pap97] S. Papert. "Why School Reform Is Impossible". In: *Journal of the Learning Sciences* 6.4 (1997), pp. 417–427 (cit. on p. 22).

[Pep10] D. Peplow. "'Keep scopin til you hear me, words is spoken clearly': Hip-hop music and the art of exclusion". In: *Working with English: Medieval and Modern Language, Literature and Drama* 6 (2010), pp. 14–44 (cit. on p. 74).

[Pia64] J. Piaget. "Development and Learning". In: *Piaget Rediscovered - Selected Papers From a Report of the Conference of Cognitive Studies and Curriculum Development (March 1964)*. Ed. by R. E. Ripple and V. N. Rockcastle. Itahca, NY: Cornell University, 1964 (cit. on pp. 38, 74).

[RCK10] C. Riegle-Crumb and B. King. "Questioning a White Male Advantage in STEM: Examining Disparities in College Major by Gender and Race/Ethnicity". In: *Educational Researcher* 39.9 (2010), pp. 656–664 (cit. on p. 31).

[Red+13] K. Redmond, S. Evans, and M. Sahami. "A Large-scale Quantitative Study of Women in Computer Science at Stanford University". In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE '13. Denver, Colorado, USA: ACM, 2013, pp. 439–444 (cit. on p. 133).

[Rei+01] B. J. Reiser, B. K. Smith, I. Tabak, F. Steinmuller, W. A. Sandoval, and A. J. Leone. "Cognition and Instruction: Twenty-five Years of Progress". In: ed. by S. M. Carver and D. Klahr. Mahwah, NJ: Lawrence Erlbaum Associates, 2001. Chap. BGuILE: Strategic and Conceptual Scaffolds for Scientific Inquiry in Biology Class- rooms. Pp. 263–305 (cit. on p. 69).

[Res+09] M. Resnick, J. Maloney, A. M. Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. "Scratch : Programming for Everyone". In: *Communications of the ACM* 52.11 (2009), pp. 60–67 (cit. on p. 10).

[Res12] M. Resnick. "Reviving Papert's Dream". In: *Educational Technology* 52.4 (2012), pp. 42 –46 (cit. on p. 12).

[RH01] P. V. Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. 2001 (cit. on p. 68).

[Ril05] A. Riley. *Who Says a Woman Can't Be Einstein?* Print. 2005 (cit. on p. 37).

[SA95] C. M. Steele and J. Aronson. "Stereotype Threat and the Intellectual Test Performance of African Americans". In: *Journal of Personality and Social Psychology* 69.5 (1995), pp. 797–811 (cit. on p. 25).

[San05] W. A. Sandoval. "Understanding Students' Practical Epistemologies and their Influence on Learning through Inquiry". In: *Science Education* 89.4 (2005), pp. 634–656 (cit. on p. 69).

[Saw+06] R. K. Sawyer, A. Collins, J. Confrey, J. L. Kolodner, and M. Scardamalia. "Moving Forward: The Learning Sciences and the Future of Education". In: *Proceedings of the 7th International Conference on Learning Sciences*. ICLS '06. Bloomington, Indiana: International Society of the Learning Sciences, 2006, pp. 1084–1087 (cit. on p. 18).

[SH99] D. Sperber and L. Hirschfeld. "Culture, Cognition, and Evolution". In: *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*. Ed. by R. Wilson and F. Keil. The MIT Press, 1999, pp. cxi–cxxxii (cit. on pp. 13, 79).

[SK07] C. Schulte and M. Knobelsdorf. "Attitudes towards Computer Science-Computing Experiences as a Starting Point and Barrier to Computer Science". In: *Proceedings of the third international workshop on Computing education research - ICER '07* (2007), p. 27 (cit. on p. 34).

[Smi15] M. Smith. "Megan Smith, Chief Technology Officer of the United States, on Data, Innovation, and Women in Technology." In: *Charlie Rose* (2015). URL: http://www.charlierose.com/watch/60554078 (cit. on p. 34).

[Ste97] C. M. Steele. "A Threat in the Air: How Stereotypes Shape Intellectual Identity and Performance." In: *American Psychologist* 52.6 (June 1997), pp. 613–629 (cit. on pp. 25, 27).

[TH12] N. Titterton and K. Haynie. "Frabjous Evaluation Survey: Attitudes Towards Computer Science". Developed as part of Frabjous Research. 2012 (cit. on p. 45).

[Toy11a] K. Toyama. "Technology As Amplifier in International Development". In: *Proceedings of the 2011 iConference*. iConference '11. Seattle, Washington, USA: ACM, 2011, pp. 75–82 (cit. on p. 23).

[Toy11b] K. Toyama. *There Are No Technology Shortcuts to Good Education*. 2011. URL: http://edutechdebate.org/ict-in-schools/there-are-no-technology-shortcuts-to-good-education/ (cit. on p. 40).

[Tra+12] E. M. Trauth, C. Cain, K. D. Joshi, L. Kvasny, and K. Booth. "Understanding underrepresentation in IT through intersectionality". In: *Proceedings of the 2012 iConference*. iConference '12. Toronto, Ontario, Canada: ACM, 2012, pp. 56–62 (cit. on p. 1).

[Tre92] U. Treisman. "Studying Students Studying Calculus: A Look at the Lives of Minority Mathematics Students in College". In: *The College Mathematics Journal* 23.5 (1992), pp. 362–371 (cit. on pp. 26, 153).

[Wal15] E. Wald. *Talking 'Bout Your Mama: The Dozens, Snaps, and the Deep Roots of Rap*. 2015. URL: http://www.elijahwald.com/dozens.html (cit. on p. 31).

[Wat13] A. Watters. *Visiting Seymour*. 2013. URL: http://hackeducation.com/2013/07/30/visiting-seymour/ (cit. on p. 22).

[WF98] B. White and J. Frederiksen. "Inquiry, Modeling, and Metacognition: Making Science Accessible to All Students". In: *Cognition and Instruction* 16.1 (Mar. 1998), pp. 3–118 (cit. on p. 70).

[Win06] J. M. Wing. "Computational thinking". In: *Communications of the ACM* 49.3 (Oct. 2006), pp. 33–35 (cit. on p. 19).

[Win08] J. M. Wing. "Computational Thinking and Thinking about Computing." In: *Philosophical Transactions of The Royal Society* 366.1881 (2008), pp. 3717–3725 (cit. on p. 18).

[Yos05] T. J. Yosso. "Whose culture has capital? A critical race theory discussion of community cultural wealth". In: *Race Ethnicity and Education* 8.1 (2005), pp. 69–91 (cit. on p. 13).

[Zwe10] S. Zweben. *2008-2009 Taulbee Survey*. 2010 (cit. on p. 1).

[Col14] College Board. *AP CS Principles Curriculum Framework*. Tech. rep. New York, NY: The College Board, 2014 (cit. on pp. 20, 21, 30, 67).

[Col15] College Board. *AP Computer Science: Principles*. 2015. URL: http://www.csprinciples.org (cit. on p. 8).

[Fiv15] Five Thirty Eight. *Police Killings - Github*. 2015. URL: https://github.com/fivethirtyeight/data/tree/master/police-killings (cit. on p. 71).

[Goo15] Google, Inc. *Made With Code*. 2015. URL: https://www.madewithcode.com/ (cit. on p. 36).

[Mak15] Makers Profile. *Katherine G. Johnson: NASA Mathematician*. 2015. URL: http://www.makers.com/katherine-g-johnson (cit. on pp. 34, 35).

[NCW14] NCWIT. *SET Award for Portrayal of a Female in Technology*. 2014. URL: https://www.ncwit.org/news/set-award-portrayal-female-technology-presented-eic-ncwit-google-honors-inspirational-potential (cit. on p. 36).

[Ret14] Retraction Watch. *The Camel doesn't have Two Humps: Programming "Aptitude Test" canned for Overzealous Conclusion*. 2014. URL: http://retractionwatch.com/2014/07/18/the-camel-doesnt-have-two-humps-programming-aptitude-test-canned-for-overzealous-conclusion/ (cit. on p. 38).

[Sec08] Secret Geek. *'Alan Kay on The Camel has Two Humps'*. 2008. URL: http://www.secretgeek.net/camel_kay (cit. on p. 38).

[Sou15] South Park Archives. *Jenkins*. 2015. URL: http://southpark.wikia.com/wiki/Jenkins (cit. on p. 33).

[The06] The Federal Bureau of Investigation. *A Byte Out of History: Taking the Pulse of Crime...76 Years and Counting*. 2006. URL: https://www.fbi.gov/news/stories/2006/june/ucr_history060706 (cit. on p. 65).

[The14] The White House, Office of the Press Secretary. "FACT SHEET: New Commitments to Support Computer Science Education". In: *The White House* (2014). URL: https://www.whitehouse.gov/the-press-office/2014/12/08/fact-sheet-new-commitments-support-computer-science-education (cit. on p. 59).

[The15] The White House. "The Untold History of Women in Science and Technology". In: *The White House* (2015). URL: https://www.whitehouse.gov/women-in-stem (cit. on p. 36).

[UC 15] UC Berkeley Office of Planning & Analysis. *Fall Enrollment Trends*. 2015. URL: http://opa.berkeley.edu/campus-data/berkeley-data-visualizations (cit. on p. 9).