

Spring 2016

Computing Language and Thinking: Analysis, Design, and Assessment of Introductory Computer Science Workshops in the Liberal Arts Experience

Kathleen Teresa Burke
Bard College

Follow this and additional works at: https://digitalcommons.bard.edu/senproj_s2016



Part of the [Computer Sciences Commons](#), [Curriculum and Instruction Commons](#), [Digital Humanities Commons](#), [Other Education Commons](#), and the [Science and Mathematics Education Commons](#)



This work is licensed under a [Creative Commons Attribution-NonCommercial-No Derivative Works 4.0 License](#).

Recommended Citation

Burke, Kathleen Teresa, "Computing Language and Thinking: Analysis, Design, and Assessment of Introductory Computer Science Workshops in the Liberal Arts Experience" (2016). *Senior Projects Spring 2016*. 220.

https://digitalcommons.bard.edu/senproj_s2016/220

This Open Access work is protected by copyright and/or related rights. It has been provided to you by Bard College's Stevenson Library with permission from the rights-holder(s). You are free to use this work in any way that is permitted by the copyright and related rights. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. For more information, please contact digitalcommons@bard.edu.

Computing Language and Thinking:
Analysis, Design, and Assessment of
Introductory Computer Science
Workshops in the Liberal Arts Experience

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Kathleen Burke

Annandale-on-Hudson, New York
May, 2016

Abstract

This project seeks to assess and improve upon a new required introductory computer science workshop for first year students at Bard College. It addresses the design and implementation of the course itself, along with the improvements needed in order to continue the program. Many students are not offered computer science courses prior to college; this program has been designed to remedy that by requiring all students to learn key concepts in computer science as a part of their orientation. The program consists of a 90 minute lesson taught by professors with expertise in fields outside of computer science, in addition to student led two-hour "coding studios" in Graphics, Robotics and Digital Literature.

Participants took a survey based on the Computing Attitudes Survey before and after the program. These surveys were paired using anonymous identification numbers unique to each participating student. This paired data was then analyzed and used to identify changing attitudes towards computer science concepts and themes. Using these results, a new and improved curriculum is designed to be implemented in following years.

Contents

Abstract	1
Dedication	4
Acknowledgments	5
1 Background	6
1.1 A Brief History of Computer Science Education	7
1.1.1 Current State of CS	7
1.1.2 K-12 Public School Initiatives	8
1.1.3 Private Organizations	9
1.2 Computer Science at Bard College	9
1.2.1 Computer Science at Bard College Today	9
1.3 Motivation	10
1.3.1 First Year Experience	11
1.3.2 Goals	11
2 Course Design and Implementation	13
2.1 In Class HTML Workshop	14
2.1.1 Required Activity	15
2.1.2 Optional Activity	18
2.2 Coding Studios	20
2.2.1 Digital Literature	21
2.2.2 Robotics	23
2.2.3 Graphics	25
2.3 Training	26
2.3.1 Language and Thinking Professors	27

<i>Contents</i>	3
2.3.2 Computer Science Tutors	27
2.4 Surveys	27
3 Results	29
3.1 Statistical Analysis	30
3.2 Anecdotal Evidence/Observations	34
3.3 Conclusions	36
4 Redesign	38
4.1 Lesson Plans	38
4.1.1 HTML Activity	38
4.1.2 Coding Studios	40
4.2 Survey	43
5 Conclusion	45
5.1 Final Discussion	45
5.2 Future Works	46
Appendix	48
5.3 Survey Demographics	48
5.4 Survey Questions	51
5.5 Additional Questions	59
5.6 Surveys	62
5.6.1 Old Surveys	62
5.6.2 New Surveys	68
5.7 Lesson Plans	70
Bibliography	77

Dedication

Dedicated to my grandparents, Gerald and Barbara Jones, and John and Mona Burke.

Acknowledgments

I would like to thank everyone involved in my journey at Bard. A huge thanks to my senior project adviser, Keith, and my academic adviser, Sven. Both of you have given me wonderful guidance since freshman year, and have encouraged me to seek out opportunities to teach. Much love and thanks to Mom, Dad, Declan, Olivia, and Jack - I would not have been able to graduate without your support. Thank you to all of my friends, for all of the laughs and late night study sessions.

Another huge round of thanks goes out to the wonderful faculty and staff of Bard College. All of my professors have been instrumental to my growth as a student and as a person. A huge thanks to Nicole Roberts, my boss since my first semester and the best boss I've ever had. Thank you to Mary Ann, Timand and Bethany from the Dean of Students office. Thank you Siira, and the Center for Civic Engagement for helping me to grow my skills as a computer science teacher. I'd also like to thank Bard Security for not kicking me out of RKC at midnight while I worked past midnight on many an assignment.

I don't know what I was thinking when I decided to start a club in my senior year, but my co-club heads for the Women in S.T.E.M. group have helped the club become a huge success. Thank you Shar, Marley and Alexandra for making it so fun and easy to support and celebrate women in S.T.E.M.! Lastly, a huge thanks to Mr. Patel at Quakertown High School, for making CS1 and CS2 so much fun and making me want to major in computer science.

1

Background

Computer science education is becoming increasingly important as the influence of computing and technology expands. Regardless of academic interests, every student can benefit from coding skills. Their daily life and job opportunities have become enveloped by coding. However, the need for literacy in computer science is not reflected in the general education classes required at most colleges, and often is not reflected in public K-12 schools.

Public perception of the typical computer scientist is warped by TV shows like *Mr. Robot* and *Silicon Valley*, along with news stories about hackers. The media tends to portray programmers and software developers as outlaws and geniuses. As a result, many students are intimidated by the subject matter. Even when computer science classes are offered, these students might be hesitant to take them. This project seeks to explore computer science education in the context of a mandatory liberal arts program for first year students at Bard College. The majority (59.8%) of these students did not receive computer science education prior to arriving at Bard College, and of those with a prospective major, very few (just 5 students) have computer science in mind.

1.1 A Brief History of Computer Science Education

The start of computer science can be traced back to 19th century France, and Charles Babbage and Ada Lovelace's work on the Analytical Engine. Ada Lovelace's translation notes on the engine are generally accepted as the first conception of a computer program. [15] However, it took much longer for computer science to become a field of study, with Maurice Wilkes establishing the first official computer science program in 1953 at Cambridge University in Britain. [3]

In 1962, Purdue University founded their Department of Computer Science, creating the first computer science majors in the United States. [13] Shortly thereafter, many other colleges and universities began to incorporate computer science courses into their mathematics curricula and eventually constructing their own computer science departments. At this stage, computer science was an academic field exclusive to a limited amount of colleges, and was not taught at a high school level.

In the 1980s, Seymour Papert became interested in teaching computer science to children as a method for confronting difficulties in math. In Seymour Papert's *Mindstorms: Children, Computers, and Powerful Ideas*, Papert refers to this struggle as "Mathophobia" and presents a solution: "Mathland," an imaginary place where Math is thought of as a natural part of a student's vocabulary and is not disassociated from the humanities and sciences. [19] Computers have a unique ability to represent mathematical concepts in real time through the use of screens and robotics. This allows students to take concepts from their math lessons, and see them interact with the real world. Papert's ideas are still utilized in classrooms today, with the use of robots to teach computer science lessons.

1.1.1 Current State of CS

In 2011, about 2,100 out of the 42,000 high schools in the United States were certified to teach the AP Computer Science Exam. [5] More recently, in 2015, about 48,000 students

took the AP Computer Science Exam in the 4,310 schools that offer it. For comparison, over 527,000 students took the English Language and Composition AP Exam in 12,760 schools. [2] Despite the fact that "computing makes up 2/3 of projected new jobs in STEM," and "computing occupations are among the highest-paying jobs" for those that graduate in computer science, not nearly enough students are graduating in time to fill these highly paying positions. [9] Some of this may have to do with the low amount of schools offering computer science. In addition, female, black and hispanic students are largely unrepresented in this field. [5]

Students who are underrepresented in computer science need support and access to computing concepts through their public school education and through inventive after school programs in order to succeed. These same students can then serve to fill high-paying, tech-related positions.

1.1.2 K-12 Public School Initiatives

According to studies by *Google* and *The College Board*, within 10 years there will be roughly a million more jobs than graduates in computer science, with only 10 percent of K-12 schools teaching computer science currently. [9] This sounds wonderful for students who aspire to find jobs in computing, but if there aren't enough graduates to fill the jobs, it will leave a huge gap in the job market. As mentioned earlier, an easy way to fill this gap is to utilize underrepresented populations in technology and help them get degrees in computer science.

In an effort to improve the amount of minority students engaged in computer science classes in public schools, President Obama's *Computer Science For All* initiative is providing funding for CS courses in public high schools as well as elementary and middle schools. [9] Additionally, many local and state governments are increasing funding for computer science classes and computer science educators. For example, *The New York City Foun-*

Initiative for Computer Science Education (CSNYC) has a 10 year plan to bring computer science education to every public high school in the City of New York. [6]

Through these local and national programs, more students are getting access to computer science education, and more students will readily find jobs that are desperately needed.

1.1.3 Private Organizations

However, in the meantime, many private and nonprofit organizations are working to teach students in after-school and summer camp programs. Many private nonprofit organizations, like *All Star Code* and *Girls Who Code*, seek to reach out to children who don't have the advantage of learning computer science in their public school classrooms. [12] [8] Additionally, events like *Code.org's Hour of Code* encourage students of all ages to engage in a short coding activity to learn a bit more about the technology they use every day, or to start their own career in computer science. [18] Through these and many other efforts taking place throughout the United States, computer science is becoming available to under-served populations.

1.2 Computer Science at Bard College

As Bard College's Computer Science Department continues to expand, courses are becoming more diverse in subject matter, and non-majors courses are offered almost every semester.

1.2.1 Computer Science at Bard College Today

Currently, the Computer Science Department at Bard College is growing, thanks to a large amount of sign-ups for introductory courses like *Object-Oriented Programming with Robots*. Additionally, computer science classes help to fill a distribution requirement in Mathematics and Computation, which encourages students to take computer science. All

mathematics majors are also required to take *Object-Oriented Programming* in order to moderate, a necessary step in pursuing their degree.

Despite the wide selection of computer science courses offered, few non-majors register for these classes. Those who do tend to register for an introductory course that relates to their academic interest. For example, 100-level courses like "Intro to Computing: Digital Humanities," "(De-)Coding the Drone," and cross-listed courses like "Foundations of Mind, Brain & Behavior" serve to meet non-majors on their own level of interest. These courses are context-based, allowing students to apply their learning to real life situations rather than just learn the theory behind computer science. This is just one way to reach out to students, but as it's not required for all students, not all students take advantage of the program.

1.3 Motivation

As most high schools do not offer computing courses, the majority of students entering Bard College do not enter with any knowledge of computer science, let alone interest in taking a course. Considering this and the fact that there is growing demand for jobs in computing, as well as programming skills applied to other fields, introductory coding makes sense to incorporate as a part of a liberal arts experience.

The growth of the Experimental Humanities program at Bard demonstrates that fields that are not traditionally linked to mathematics and computing can benefit from their influence. Courses like *Introduction to Media*, *Cybergraphics*, and *History of Experiment* combine literature, studio arts and history with coding assignments. This enhances the learning process and allows students to make new discoveries.

Based on these observations, it makes sense to prepare students to utilize coding ideas from their very first weeks at Bard.

1.3.1 *First Year Experience*

First year students at Bard College start their experience with a three week course entitled "Language and Thinking." Taught largely by humanities professors from other universities, this program introduces students to writing at a college level. They write informed responses to challenging academic material and tackle difficult topics with nuance. Most of the reading is printed material and students are required to write in a paper notebook just about every day. At first glance, this doesn't appear to be the perfect environment to teach computer science concepts.

However, computer science can be taught with or without a computer as exemplified by Ada Lovelace's visionary notes on the analytical engine in 1842. Despite the fact that the Analytical Engine was neither complete, nor in her possession, Lovelace noted that "the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent." Selected materials in the Language and Thinking anthology include stories and poetry by Jorge Luis Borges. His writing in "The Garden of Forking Paths" and "The Library of Babel" evokes strong comparisons to the unique types of logical conundrums brought up by programming and algorithms. [14]

Clearly, computer science can be taught in the same spirit of Language and Thinking, while encouraging students to pursue an area they may not have experienced before.

1.3.2 *Goals*

The Computing Language and Thinking project aimed to convey key concepts from computer science to students through the program:

- Close-reading of source code
- Networks
- Logical structure

- Formats, Protocols and Algorithms
- Data Representation

Given just a short in-class activity and a student-led coding studio, it was key to create lessons that would be easy to learn in a short amount of time, but would still be engaging and entertaining for students. It was also important to keep in mind that most of the professors teaching would have no prior knowledge of programming or any coding languages.

As a result, one of the main goals of the Computing Language and Thinking workshops was to break down concepts to a level that would be easily taught by either the Language and Thinking professors or computer science students. Another goal was to show students that these coding concepts could apply to their own interests academic and otherwise. A useful measure of the effectiveness of a computer science course is the Computing Attitudes Survey. This survey has been used in several studies, including ones at the University of Nebraska Omaha. [4] A modification of the survey was used to measure change in student attitudes over the course of the 3 week program. With these metrics, restrictions, and goals in mind, course development began.

2

Course Design and Implementation

HTML In-Class Activity		Coding Studios		
Required Activity	Optional Activity	Digital Literature	Graphics	Robotics
90 minutes	30 minutes	2 hours	2 hours	2 hours

Figure 2.0.1. Structure of the computer science curriculum for the Language and Thinking program.

Starting in June 2016, Professors Keith O’Hara and Sven Anderson, assisted by Diana Ruggiero and myself, began creating the computing curriculum as part of the Language and Thinking Program. The aim of the curriculum was to create thought-provoking discussion, connecting the ideas of algorithms and programming to the readings used in the program, and getting each student to program for perhaps their first time.

The three week Language and Thinking program incorporated the computing portion by allowing for a 90 minute in-class activity, as well as an optional portion. Additionally, students were required to take a two hour coding studio after class. These activities all

Planning Stage			Language & Thinking Program		
Course Design	Professor Training	Student Training	Pre Surveys Distributed & Collected	Coding Studios & In-Class Lessons	Post Surveys Distributed & Collected
June and July	Mid-July	August 3-7	August 10	August 10-21	August 24, 25 (differs by instructor)

Figure 2.1.1. Timeline for the design and implementation of the course material.

drew from core texts in the Language and Thinking anthology, mainly Jorge Luis Borges' *The Garden of Forking Paths* and *As We May Think* by Vannevar Bush.

2.1 In Class HTML Workshop

Drawing from the readings in the Language and Thinking anthology, one of the first ideas involved manipulating text from the readings to create new interpretations and draw new meaning from the material. Using JSBin, we created examples of highlighting, augmenting, and modifying pieces of text from the selected readings. This idea drew inspiration from blackout poetry, wherein an artist would paint or white out portions of the page in order to create a sentence from words scattered throughout the page. Moving from paper to computer, the next step in connecting this activity to coding was to modify texts through code. From this, we worked back towards something more grounded in the familiar academic setting the Language and Thinking professors and first year students would be most familiar with.

Inspired by the idea of "unplugged" computer science activities, we decided to work the concept into each section of the curriculum. Unplugged, in this context, means recreating the ideas behind coding activities through methods that don't require a computer. For instance, the *CS Unplugged* website teaches sorting algorithms by having students sort objects by hand. [7] This tactile experience allows students and professors who might be

	JSBin	Trinket	Codepen	Student Server
Software install	None	None	None	Need FTP client
Difficulty	Low	Low - Medium	Medium	High
Sharing	Short URL	No direct links	No direct links	Personalized by student's Bard username
Accounts	Paid premium membership	Paid "connect" membership	Free	Free
Allows for private drafts	Yes; with membership	Yes; with membership	No	Yes - just save on computer
Live changes	Yes	Yes	No	No

Figure 2.1.2. Comparison of the different HTML editing tools considered for the HTML activity. Lighter cells indicate better options.

wary of coding to try it out in a non-threatening environment before typing up some code.

This was especially important to include for professors with no experience in coding.

Considering this challenge, and the interests of the professors, we decided to focus the class time on an HTML activity and an unplugged activity to demonstrate *Google PageRank*. Hypertext Markup Language (HTML) involves much less complicated syntax than most programming languages, and allows beginners to see results quickly, especially using tools like JSBin. JSBin was chosen over other educational options for a few key reasons. Having a shareable, short URL, ease of use, private draft options, and an ability to make changes live quickly were all very important in order for the activity to be a success. JSBin is able to provide all of these without having to install any software, and as such, it was the best option. [10]

2.1.1 Required Activity

The in-class portion of the Computing Language and Thinking project focused mainly on learning about HTML and Google's PageRank algorithm. Professors were required to teach a 90 minute lesson teaching HTML to students resulting in students creating their

own individual websites. Based on interest and the amount of time left in their schedule, professors could choose to supplement this activity with an optional 30 minute activity demonstrating how Google's PageRank algorithm works.

The in-class HTML activity was designed to allow students to follow along and teach themselves, with the assistance of a professor leading the class. The first portion of the required lesson presented students with a paper with text on the left side of the page, and then a blank half on the right side. Students were required to modify and change the text however they wanted on the right side of the page. Drawing, underlining, getting rid of and adding words were all encouraged. This portion ended with a discussion about how this might be possible to do through code, and what types of strategies students might use to recreate this.

Students were then invited to open the JSBin link on their computer or share with another students in order to participate. They created their own JSBin accounts and then cloned their own copy of the main page. Cloning a copy of the "bin" allowed students to edit a copy of the original code and make modifications without impacting the source bin. Following along with the instructor, students went through each activity in order, starting with Emphasis and Formatting.

Using the JSBin website, students are instructed to create an account and view the source behind the website. After analyzing and reading the code, they continue on to 4 different coding activities. The first, called "Emphasis and Formatting" asks them to create their own spin on a piece of text by changing the color and formatting of the text using the HTML code on the left side of the screen. Firstly, they write, draw, and doodle on a piece of paper with a piece of text on it, and then they use JSBin to modify the text using HTML tags. The "Emphasis and Formatting" JSBin page teaches students how to make text bold, italicized, crossed out and underlined through HTML tags. Additionally, it demonstrates how to change the font styles, colors and sizes. A short excerpt from

The screenshot displays the JSBin.com interface for the 'Emphasis & Formatting' section. The top navigation bar includes 'File', 'Add library', 'Share', 'HTML', 'CSS', 'JavaScript', 'Console', 'Output', 'Login or Register', 'Blog', and 'Help'. The main content area is split into two panes: a code editor on the left and an output preview on the right.

The code editor contains the following HTML code:

```

<html>
<head>
<title>Emphasis & Formatting</title>
</head>
<body style="color:#004242;font-family:Arial, Helvetica, sans-serif">
<center><h2><u><b>Em</b><i>phas</i><u>sis</u><small>&&</small>
<b>For</b><b>big>mat</big><i>ting</i></u></h2></center>

<div style="background-color:#80B280; color:white; padding:20px;">
<font size="4">Create your own spin on text by choosing a section of
text from Vannevar Bush's <i>As We May Think</i>. Below is a short
section from <i>As We May Think</i>, but you can feel free to choose any
text you'd like to modify.</font><br>
<p>Tagging words as <b>bold</b> or <i>italicized</i> or
<u>underlined</u> is an <strong>easy</strong> and <em>effective</em> way
to change the <b>pre</b><i>tat</i><u>ion</u> of the text.You can also add
<br>line <br>breaks,<br>change <font face="arial">font style</font>
and<font color="yellow">font color</font>, as well as the <font
size="24">size</font> and <font style = "background-
color:green">background color.</font>
<p>Redacted poetry is when words from a selected text are
<s>crossed-out</s> by hand or otherwise made <font color =
"#80B280">unreadable</font> so the remaining words say something
interesting (Examples by an artist working with newspapers <a
href="http://austinkleon.com/category/newspaper-blackout-poems/"
target="top">here</a>). </p>
<p>Alternatively, you can also use "comments," a tag that the browser
ignores. HTML comments will appear in the source code but not the output!
Using the HTML comment tag, <!--Like this!!--> comment out some of the
words in the text so something neat is left viewable! </p>
<p>
</p>
</div>
<p>

```

The output preview shows the rendered HTML, featuring a green background for the main content area. The title 'Emphasis & Formatting' is underlined and bolded. The main text is white on a green background, with various formatting examples like bold, italic, underline, font size, and background color. A redacted section of text is shown with a black background. The output also includes a link to a website and a comment tag.

The console at the bottom shows the rendered text: "A new symbolism, probably positional, must apparently precede the reduction of mathematical transformations to machine processes. Then, on beyond the strict logic of the mathematician. lies the application of logic in everyday"

Figure 2.1.3. An example of the JSBin.com layout. This screenshot is of the Emphasis and Formatting section of the HTML lesson.

the Vannevar Bush reading featured on the bottom of the page, with the instructions to change the text or add their own.

The second activity, "Hyperlinks" asks them to change the meaning of the text by adding hyperlinks to different sections of the code. They are also challenged to find out how images are added to the page by doing a close reading of the code. The "Hyperlinks" activity taught students how to link to different websites, and demonstrated how to display images that are sourced from other sites. Students are instructed to link to relevant websites based on associations they make between the text and other sources. Again, an excerpt is used from the Bush reading, and can be modified however students choose.

The next section, "Images and Multimedia," demonstrated multiple ways to modify images, as well as animated GIFs, videos, and embedded objects like songs from other websites. At this point, students are encouraged to create their own page from scratch by

opening JSBin.com in a separate tab and typing in the correct tags and links to include images and different embedded multimedia. The images and media featured on the original page feature references to the Memex from Bush's "As We May Think." [20]

The fourth activity is "Code Surgery," a short page featuring multiple mistakes and errors. By taking a close look at the HTML, one can discern that a list has been broken up into pieces, an image isn't being featured, and a link isn't leading anywhere. Given an example of what the website should look like, students are allowed at least 10 minutes to change their code to make the output match the example. Afterwards, students are encouraged to look at each others solutions.

The last coding assignment they participate in is an "HTML Freewrite." They can use any of the skills they've learned in the previous activities to create their own website. Similar to the Language and Thinking class structure, they're provided with a blank slate to start with, and given a short amount of time to express themselves however they see fit. With the additional ability to feature images, music, and videos in their free-write, this allowed for a wide variety of websites. Once their website is complete, they're asked to write down their URL on a piece of paper, and they pass them back to the instructor.

2.1.2 Optional Activity

The optional activity builds on the JSBin free writing activity, asking students to link to each others websites in order to create a small network. The instructor uses the pieces of paper with URLs to each student website and puts them in a hat, box, or bag. Each student selects a piece of paper, writes down the URL, and then puts the piece of paper back into the container. This allows for each site to have an equal opportunity to be selected, and certain sites would get selected more often than others. After each student has a new URL, they go back to JSBin and open up their website. They add a link to the URL they wrote down, and then participate in an unplugged activity.

In order to visualize the network created by these links, one student is asked to take a ball of yarn, and throw it to the person their link connected to, while holding on to one end of the yarn. The person in possession of the ball of yarn would then throw the yarn to the person they linked to, and so on. This allows them to visualize the web of links they just created. When students were thrown the yarn after already having linked to someone, they would cut off the end of the yarn and give the ball to a student who wasn't involved in the "web" of yarn yet. This continues until all students are connected, and then the professor would instruct them to try to untangle the network until the structure seemed a little less complicated. Students are invited to guess who might be the strongest, or most powerful website based on the linking structure in place.

Students are then provided with 12 wooden nickels to serve as a representation of the value assigned to each website. They then participate in three rounds of distributing the power according to a basic version of Google's PageRank algorithm. Each round is composed of three parts - passing tokens to their linked site, collecting tax, and then redistributing this tax.

- STEP 1: Passing Tokens

Each student must pass all of their tokens to the website that they linked to - regardless of how many tokens they have. Once this happens, many students are left without any tokens, and therefore their links would not have power until they are given new tokens.

- STEP 2: Tax

The instructor collects roughly 1/4 of the tokens from each person, or "site."

- STEP 3: Redistribution

The instructor then breaks the collected tokens into piles of 3, one for each person

representing a site. Each person is given 3 tokens, regardless of their current amount of tokens.

After 4 or 5 rounds, the instructor and students participate in a discussion of why this activity connects to their class and to computer science. A reading about Google PageRank is an optional part of this discussion. They also are encouraged to discuss which students have the most power based on the activity.

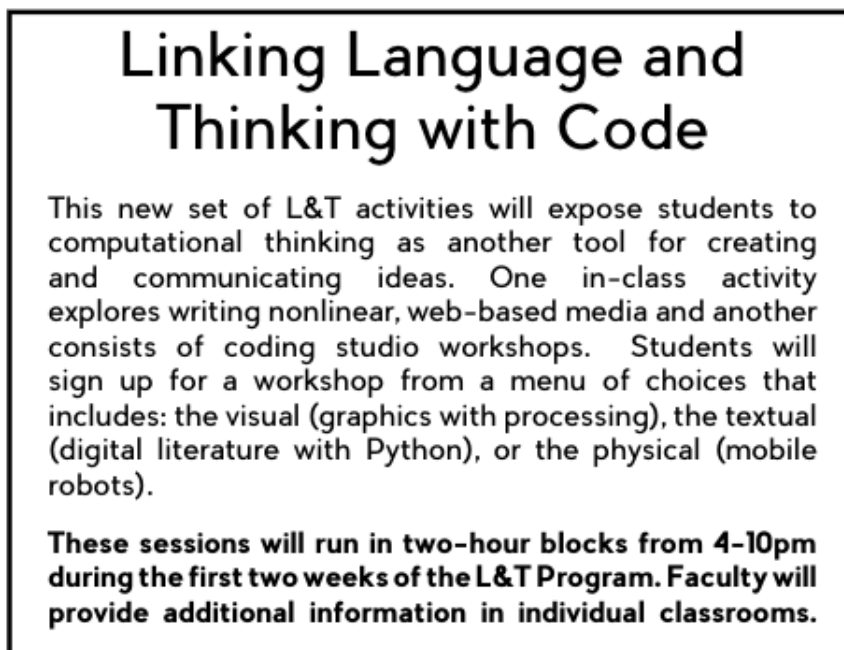
The benefits of the first day activity is that it allows students to interact with code in an environment and manner that is similar to their Language and Thinking or literature classes. They employ close reading techniques, participate in a freewrite, and learn some HTML along the way. The second day activity allows them to think about the power structures on the internet that allow them to find information that is relevant and reliable.

2.2 Coding Studios

The coding studios were designed to allow computer science students to teach first year students about more complicated concepts such as loops, functions and libraries. Taught in Python and Processing, these languages allowed students to read the code easily, learn quickly, and modify the code without having to learn any over-complicated syntax.

Another key part of the coding studio was the ability for students to choose the activity they were most interested in. If a studio art major was wary of coding, perhaps the graphics studio would interest them, and if a student wanted to stick to literature rather than robots, they could select the Digital Literature coding studio. The Coding Studio was a required activity, but the diversity of topics served differing interests.

The coding studios were advertised through posters, Facebook posts, announcements during Language and Thinking day classes, emails to the First Year listserv, and a notice in the program given to students at the start of orientation. Links were given to a sign up



Linking Language and Thinking with Code

This new set of L&T activities will expose students to computational thinking as another tool for creating and communicating ideas. One in-class activity explores writing nonlinear, web-based media and another consists of coding studio workshops. Students will sign up for a workshop from a menu of choices that includes: the visual (graphics with processing), the textual (digital literature with Python), or the physical (mobile robots).

These sessions will run in two-hour blocks from 4-10pm during the first two weeks of the L&T Program. Faculty will provide additional information in individual classrooms.

Figure 2.2.1. The description of the coding studios in the Language and Thinking program given to students.

page on Doodle.com, allowing 10 students to sign up per 2 hour session. 30 sessions were offered per coding studio over the course of 2 weeks, allowing for 90 options for students. As a result, more than 400 students participated in the coding studios, and multiple students participated in more than one studio. These 2 hour workshops allowed students the ability to connect computer programming to other disciplines and interests.

2.2.1 Digital Literature

The Digital Literature coding studio was advertised as an activity wherein students would learn to "read and write algorithms that generate poetry and prose." The resulting workshop appealed to students with an interest in literature or poetry.

This coding studio focused on teaching Python to students interested in how it might allow for the analysis of text. The "unplugged" activity involved filling out mad libs using lists of parts of speech written on the board by students. This helped to demonstrate

how random choice and lists impact the way the mad libs would turn out. Following this, students read out the results of their mad libs and discuss how the activity might relate to coding. Oftentimes the students would come up with the word "randomness" on their own, but if the discussion didn't touch on that, instructors would bring it up.

Using iPython notebooks, instructors would lead students through a short coding "boot camp" teaching them how to use lists, print strings, create functions, and use loops. [1] The students would get a chance to try each of the examples on their own computers, using a blank iPython notebook. Each example became more complex.

Once students had a rough idea of how programs in Python worked, the instructor had students open up a pre-written iPython notebook containing code that produces a short love letter, based off of Christopher Strachey's original love letter generator, written in 1952. [17]

Students were encouraged to perform a "close reading" similar to what they would be doing in their Language and Thinking classes. After a few minutes, the instructor walks them through each line of the code, asking for students to speak up if any line was confusing, or if they wanted to make a guess as to what the code was doing. Once the students understood where the text was coming from, they were encouraged to change the words in the lists for each part of speech, however they wanted to.

By changing the words in the list of adjectives and salutations, students could get the code to generate wildly differing letters. Additionally, they were encouraged to change the Python code outside of the lists of strings. This allowed for even more variation, and depending on the skill level of students, encouraged students to challenge themselves.

Instructors gave examples of previous modifications of the code, including hate letters, complaint letters, and love letters with abbreviations and texting lingo. More advanced students were encouraged to change the text that created the structure of the sentences,

KIND SUBORDINATE,

I RESPECT YOUR SYNERGY!
MY SERIOUS SYNERGY SYNERGISTICALLY PROMOTES YOUR CAREFUL HONESTY.
PLEASE CONTINUE YOUR DUTIFUL TRUTHFULNESS.
MY CAREFUL HONESTY TRUTHFULLY ACTS IN FAVOR OF YOUR SERIOUS HONESTY.
MY TRUTHFUL DIRECTNESS CAREFULLY RESPECTS YOUR CAREFUL CAREFULNESS.

BEST WISHES CAREFULLY,

MANAGEMENT

Figure 2.2.2. An example of the a letter resulting a new take on the code.

in order to make more significant changes. Many classes would dissolve into giggles by the end of this activity.

This coding studio's strength lied in the simplicity of the lesson, the interactivity between the teaching and the actual coding, and the fun side of comparing coding to mad libs.

2.2.2 Robotics

The robotics coding studio was based mainly on activities used in CMSC 143: Object Oriented Programming with Robots, a course typically taught to freshmen intending to moderate into the Computer Science program at Bard College. This course utilizes the Scribbler 2 robots and IPRE Calico runtime environment to teach students Python and object-oriented programming.

The robotics coding studio is broken into 3 parts: using the game controller to draw, learning the key commands for movement, and creating a song and dance. This allowed students to become familiar with the robot before being asked to look at code simply through the screen.

Drawing with the robot was initialized using the Myro module in Calico. Students type in the command "gamepad()" and then place a marker into the hole in the center of the Scribbler robot. Then, by manipulating the controls on the controller, students could get their robot to draw different shapes. The instructor would encourage them to think about strategies for drawing rectangles and stars, and then ask them to turn off the gamepad function, in order to learn how to program a short code to make the robot move and draw the desired shape.



The instructor introduces different commands to get the robot to move forward, backward, and turn, and asks students to try them out on their own computers. Then, using trial and error, students would write their own code to draw a shape without using the gamepad function. This allowed students the opportunity to debug and modify their code, and see the results instantly.

Following this activity, students were taught how to make their robot emit different beeps. The instructor demonstrated these functions and then showed an example of a song and dance written for the robot. Students were encouraged to come up with their own original song and dance, and given the rest of the class time to play around with it. At the end, all students were given the chance to show off their code to the rest of the class.

The strength in this activity was that it allowed students to interact with a physical object instead of simply watching pixels on the screen. It appealed to tactile learners and stressed the ability of computers to influence the world around them. Through the activity, students learned how to write a function, perform for-loops, and debug and modify code that interacted with the real world.

2.2.3 Graphics

The Graphics coding studio reached out to students that were interested in visual arts and animation. Creating filters for images, drawing shapes and changing behaviors of the shapes as they were animated all were crucial parts of the studio.

The instructors would begin by showing students the end result of the project they were about to complete and then leading them through the activity. According to the instructors, students responded well to this approach, and were interested in getting to learn the process behind filters on images and animation.

The first activity involved animating fish in a fish tank. Again, students would open pre-written code and modify it to their liking. Instructors showed off their own creations and previous versions of the fish tank in order to encourage creativity and challenge the students to create something better.

With the filters, students were introduced to the concept through pre-written code that instructors would walk them through. The code demonstrated for-loops and while loops, and instructors visually demonstrated the iteration by drawing a small graph on the board with "pixels" that would each get changed as the loop continued. Demonstrating a few modifications to the code, the main instructor would encourage students to change the code with a goal in mind - perhaps to make everything have a slightly reddish tint, or to create a black and white version of the current image.

Planning Stage			Language & Thinking Program		
Course Design	Professor Training	Student Training	Pre Surveys Distributed & Collected	Coding Studios & In-Class Lessons	Post Surveys Distributed & Collected
June and July	Mid-July	August 3-7	August 10	August 10-21	August 24, 25 (differs by instructor)

Figure 2.3.1. Timeline for the design and implementation of the course material.

At the end of the class, students would send in their fish animation code to the instructor, and while one instructor discussed other projects that might be of interest, the second instructor would put together the completed fish tank, and display it on the projector for all the students to look at.

Each lesson ended with a discussion of what projects students might be interested in tackling through the use of Processing, and some feedback about the activity.

2.3 Training

The training was conducted in two parts. The first portion of training took place in mid August, with Professors O’Hara and Anderson leading two classes of Language and Thinking Professors through the HTML activity. They were informed that computer science students would be on hand to help them if necessary, and that a hotline would be available to call.

The second part of training took place a few weeks later, as a week-long group discussion and run-through of the coding studio lessons and the in class activity. Three students were designated to teach each coding studio, with one student in each group taking the role of the lead teacher. This allowed for the other two teaching assistants to interact more personally with the first year students and troubleshoot any problems.

2.3.1 Language and Thinking Professors

Professors Anderson and O'Hara split the group of Language and Thinking professors into two groups, and led them through the HTML activity, from creating their JSBin accounts to conducting the PageRank activity complete with yarn and wooden nickels. Multiple spots of confusion and conflict were resolved during this session, and many notes were taken for improving the websites.

It also became evident that some professors would need an additional hand in the classroom in order to effectively teach the lesson. Professors were advised that a student hotline would be available to call if necessary, and that there would be a teaching guide version of the websites sent out to them over the weekend.

2.3.2 Computer Science Tutors

Computer science tutors were used as a kind of focus group for the computing activities, and gave lots of useful feedback for each of the three coding studios, resulting in improved curriculum for the actual classes. Three students were selected for each coding studio and students signed up for hotline shifts.

By the end of the week, lesson plans were distributed, surveys and handouts were printed, and sign ups for the coding studios were online.

2.4 Surveys

The surveys served to give feedback about students attitudes toward computer science before and after participating in these activities. Basing many questions off of the Computing Attitudes survey utilized in previous studies on college and university level computer science courses, the survey used robust Likert-type questions to determine student opinion. [4] [21] Surveys used in this study were approved by Bard College's Institutional Review Board. All participants were given consent forms along with their surveys.

Pre-surveys were distributed by Professors at the start of the Language and Thinking program, along with consent forms. Students were free to decide not to participate either before or at any point while filling out the survey. Surveys were collected at the end of the Language and Thinking program by professors and then returned to Professors O'Hara and Anderson.

3

Results

While the survey was our primary source for assessing the impact of this program, much was learned in the process of teaching the course and getting feedback from professors and students. Although these are not the most reliable of sources, there were issues with timing and communication that can't be observed through survey or any type of diagnostic.

There were 391 responses for the entry surveys and 368 responses for the post surveys. 264 of the surveys had both a section id and a survey identification number, allowing for paired surveys. Of those paired surveys, 109 were male and 149 were female.

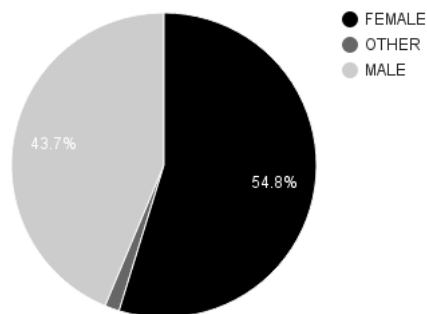


Figure 3.0.1. Genders of Students involved in the Language and Thinking program.

3.1 Statistical Analysis

Surveys were based roughly on the Computing Attitudes Survey (CAS) created by Brian Dorn and Allison Elliot Tew at the University of Nebraska Omaha. This survey asks participants to rate statements from Strongly Disagree to Strongly Agree. This survey is designed to determine the impact of semester-long courses in computer science. Through histograms and paired t-tests, it became evident which questions had changed over the course of the program, and which ones had changed significantly.

For each statement, the R programming language conducted paired t-tests for the paired surveys taken during the Language and Thinking program. A *p-value* below 0.05 indicated a significant difference between the initial results and the post-surveys. The first statement, "I think about the computer science I experience in everyday life," had no significant difference between the first and second survey. All this indicates is that over the course of 3 weeks, the program failed to change this for the average student.

Q1: I think about the computer science I experience in everyday life.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-0.55132	0.5819	-0.13852868	0.07792262	-0.03030303	0.03393166

Q2: Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business)					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
3.4497	0.0006535	0.06503319	0.237997128	0.1515152	0.2123146

Q3: I find the challenge of solving computer science problems motivating.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-2.9431	0.00354	-0.27817286	-0.05516047	-0.1666667	0.1811336

However, the second statement "Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art business)" had a small impact, indicated by the *p-value* and *Cohen's d* values. With a *p-value* below 0.05, this means that there has been a significant difference between the first set of data and the second set of data. The *Cohen's d* value indicates the effect size for the statement. Since the *Cohen's d* for this statement is between 0.2 and 0.5, the effect size is considered to be small. For this

statement, there was a positive change, indicating that more students agreed with this statement after the program than had previously. This is a great sign, as it indicates that students may have held preconceived notions that computer science might not be useful for other fields of study, but after taking the course, recognized that techniques from computer science could apply in more than just computer science. However, it's good to keep in mind that this was a small effect.

Statement three, "I find the challenge of solving computer science problems motivating," found a low *p-value* and a small effect size, along with a negative change. This indicates that students participating in the Language and Thinking program were slightly less likely to agree with the statement after taking the computing classes. It's difficult to pinpoint where this change might have come from, but it indicates that students were less motivated to solve computer science problems after taking the course.

Q4: I enjoy solving computer science problems.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-5.0665	7.64e-07	-0.3839788	-0.1690515	-0.2765152	0.311821

The fourth statement has our lowest *p-value* and highest *Cohen's d* values of all the statements so far, indicating that there was the largest change in opinion for the following statement: "I enjoy solving computer science problems." This is concerning, because there was a negative difference between the pre- and post- surveys. Students might have responded this way because many had not experienced a computer science class prior to arriving at Bard College. In order to test this further, I ran the paired t-test for those who had experienced more than one course in computer science and then another for those who had never experienced computer science.

Statement five had a significant difference, but an very low effect size. This statement had a positive difference, indicating that participants tended to agree with the statement, "Reasoning skills used to understand computer science can be helpful to me in my everyday life," more in the post-survey. This did have a very low effect size, but may lead one to

believe that students might have understood the importance of computer science as a result of the program.

Q5: Reasoning skills used to understand computer science can be helpful to me in my everyday life.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
2.2976	0.02237	0.01679089	0.21805760	0.1174242	0.1414051

Q6: Learning computer science is just learning how to program in different languages.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-2.9903	0.003051	-0.27012970	-0.05562787	-0.1628788	0.1840402

The post survey revealed a decrease in participants agreeing with the statement "Learning computer science is just learning how to program in different languages." With a low *p-value* and a small effect size, it doesn't mean that there was a huge leap in students disagreeing with the statement, but it was encouraging to see that this suggests participants might have changed their opinion about the challenges involved in computer science beyond just learning the syntax for different languages.

Q7: The subject of computer science has little relation to what I experience in the real world.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-2.5446	0.01151	-0.27547772	-0.03512834	-0.155303	0.1566088

Q8: I am interested in learning more about computer science.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
1.9538	0.05178	-0.0007945371	0.2053399916	0.1022727	0.1202509

Statement seven, "The subject of computer science has little relation to what I experience in the real world," had a significant difference, but the effect size was too small to really say that there was it had a meaningful effect on the students. This indicates that there were less students agreeing with that statement, but not enough to say that it made a real impact. However, this is a good sign that students started to see a relation between their real world experience and the subject of computer science.

Question 8 had neither a small enough *p-value* or a high enough *Cohen's d* value to be able to make any conclusions. There was a slight positive increase in students agreeing

with the statement "I am interested in learning more about computer science," but not enough to make any inferences.

Lastly, there were seven questions that were only included in the post survey. These questions were:

- When I'm trying to learn something new in computer science, I find it useful to write a small program to see how it works.
- A significant problem in learning computer science is being able to memorize all the information I need to know.
- Understanding computer science basically means being able to recall something youve read or been shown.
- The readings about computing were relevant to L&T.
- The in-class computing activities were illuminating.
- I participated in the L&T coding studios.
- I see myself writing another computer program someday.

The responses to these questions were generally neutral, but the response to "A significant problem in learning computer science is being able to memorize all the information I need to know" had an average score of 3.35, with 3 being "Neutral." This indicates that students saw memorization as a roadblock to their learning of computer science. The other question that had an interesting response was "The readings about computing were relevant to L&T." Nearly no one selected "Strongly Agree," as shown in figure 3.1. This is somewhat discouraging, but to my knowledge, not all of the Language and Thinking professors utilize all of the readings in their class, and I could definitely see some students finding a disconnect between the readings and the activities.

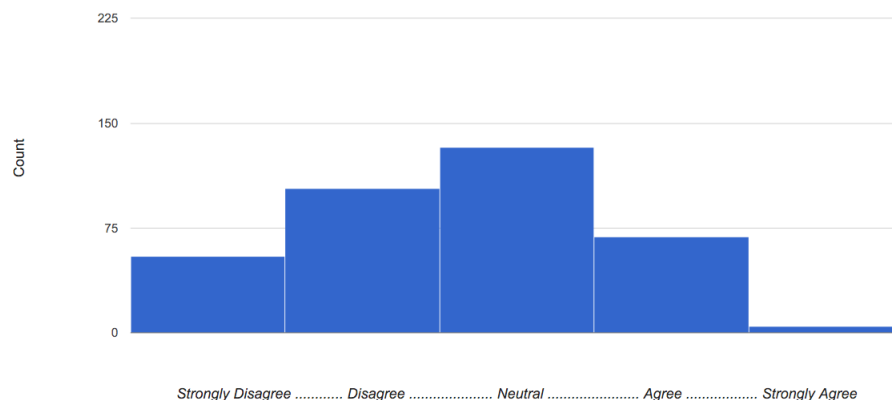


Figure 3.1.1. Q12: The readings about computing were relevant to L&T.

For more information on the survey data, see the appendix.

3.2 Anecdotal Evidence/Observations

Although these survey results are the main source for information about improvements in the course, the experience of teaching the coding studios and receiving feedback from students and professors outside of the survey was also a significant factor in the improvement of the curriculum. As such, I'd like to summarize some common concerns here.

Firstly, when training the professors, there were lots of concerns about teaching HTML to students and being able to convey the importance of the PageRank activity to students without having a background in computer science. Some professors utilized the hotline in order to contact computer science students and have them assist in the classroom. In some cases, the computer science students ended up teaching the class for the professors. One student described to me a situation where 3 different Language and Thinking classes were grouped together for the activity, and one professor taught the class, with the assistance of one CS student.

This is a huge concern, and may have led to a less-than-optimal experience for students in the classroom. Activities were designed with group of 12 students in mind, but a group

of around 36 students might not have had the ability to learn the key concepts. After speaking to a couple first year students about their in-class activity, it became evident that they preferred the experience of the coding studios to the in-class HTML activity. One student said "I wish I could have taken all three of the coding studios rather than just one."

In teaching the Digital Literature coding studio, some key changes were incorporated over the course of the two weeks. Students were encouraged to follow along with the activity while sitting a distance away from their computers, and then only to start typing when instructed. This allowed for more control over the attention of the class, and made sure that all students understood the key concepts of the lesson before delving into the modification of the love-letter code. Additionally, instructors reported that sharing the results of the letters before the end of the class encouraged students to challenge themselves. Students would ask one another what they did in order to get a certain result, and would then modify their code in order to make their results more entertaining.

The Graphics coding studio changed in order to deal with timing concerns, and based on student interest, would incorporate more complicated activities. The photo filters activity often became too difficult for students to understand, with little reward. The animated fish ended up becoming the most interesting activity, as little changes would amount to large changes in terms of the speed and interaction with the fish animation.

The Robotics coding studio went fairly smoothly, with few students becoming disengaged. Some of this may be due to the fact that the curriculum was loosely based on activities in Bard's "Object-Oriented Programming with Robots" course. As this course is one of the first courses taken by computer science students, the instructors were familiar with the material. However, some minor issues instructors ran into include needing to explain loops more fully than the lesson plan had explained. Students also tended to take

longer on the drawing stage of the activity than expected, leading to class running over time.

3.3 Conclusions

Based on the statistical analysis, it seems that the two main statements that had both statistical significance and meaningful effect size were the following:

- Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business)
- I enjoy solving computer science problems

The first statement had a positive change, indicating that participants learned that computer science techniques can apply to more disciplines than they would have previously thought. However, the second statement saw a decrease in student agreement. While this appears to indicate a decrease in enjoyment of computer science, one has to take into account the number of students who had no prior computer science experience. It's hard to figure out how they should answer given that they had no idea if they would enjoy solving computer science problems. After the Language and Thinking program was over, it would make sense that you see a decrease in students stating that they enjoy solving computer science problems, because they had limited selection of which computer science workshops they attended, and it took up two hours that would otherwise be free time.

In re-designing the lessons and surveys, it makes sense to focus on improving the number of students that would agree with the statement "I enjoy solving computer science problems," and then also to continue increasing the understanding of how applicable computer science concepts are to other disciplines. Another issue is that students' agreement with the statement "I find the challenge of solving computer science problems motivating" de-

creased over the course of the program. Motivating students to want to solve the problems in the lessons would improve the program greatly.

Additionally, the feedback from instructors, professors and students indicated that the largest problems came from the in-class HTML activity. Professors without experience in HTML or any programming languages found it difficult to teach concepts they were unfamiliar with. Compounding this issue, many students found the lesson very basic and were bored. This might explain the answers to the statement about challenges in computer science, as these students gave feedback indicating that they weren't challenged or motivated to engage with this lesson.

With these issues in mind, the redesigned lesson plans and curriculum aim to improve the experience for students and professors, and to challenge students.

4

Redesign

Looking at the results described in the previous chapter, the redesign of the program aims to fix or ameliorate any issues. As the HTML activity caused issues for both students and professors, it was the focus of the redesigned curriculum.

4.1 Lesson Plans

Generally, it seemed that enjoyment and motivation decreased over the course of the program, while the usefulness of computer science seemed to be communicated successfully. This indicates that the course would benefit from moving towards more fun and challenging lessons instead of dry or boring material. These revelations from the survey data prompted me to take a more creative approach while increasing the difficulty for all the lessons, especially the HTML activity.

4.1.1 HTML Activity

In order to alleviate any issues in the HTML activity, the newer activity has been redesigned to be taught by computer science students rather than the Language and Thinking professors. The hope is that by having students familiar with the material leading the

class, it will allow for the class to move quickly while allowing for a higher difficulty to create a challenge.

The decision comes down to whether this should be a part of the core curriculum for Language and Thinking, where all students are required to take the course during normal class hours, or if students should have the HTML activity as part of the after class coding studio. This will be determined according to the schedule and requirements determined by the director of the Language and Thinking Program for 2016. In both situations, computer science students should be teaching the lesson. If the lesson is in class with the presence of the Language and Thinking professor, this could allow for more discussion directed by the professor. This is especially important when considering the connections between the HTML activity and the readings concerning the Memex and PageRank algorithm. [20] [16]

However, if the lesson were to become a coding studio, the HTML lesson could allow for students interested in the topics to self-select based off of their own preferences. This self-selection could possibly solve the issues of participants decrease in motivation and enjoyment of computer science problems. On the other hand, students who do not choose to take this coding studio could miss out on the key lessons in the HTML lesson. The PageRank article and Memex reading served as the core texts for the computing portion of Language and Thinking program and as such, the lesson relating to it is important to convey to each student.

The problem then becomes how to provide professors with the opportunity to connect concepts from the reading to their own lessons without relying on them to teach the actual programming. As most of the training for Language and Thinking is taught by having the professors take the courses themselves, the improved activity should involve the professors learning the lesson themselves, taught by Professors O'Hara and Anderson, and then have the CS students teach the first year students. This will allow for the Language and

Thinking professors to have a solid grasp on the concepts without the pressure of having to teach it themselves.

Additional changes made to the lesson plan include modifying the activities so that it follows a similar format to the coding studios. Instead of leading with the coding portion of the lesson, the PageRank activity demonstrating the algorithm using yarn and wooden nickels will take precedence. This unplugged activity serves both as an ice-breaker and a motivator for the rest of the lesson. If students see the power of this algorithm while getting up and moving around and interacting with each other, it can motivate them to make a website that shows them how to create those links.

The coding portion of the lesson has also been changed to involve less complication. Students will not need to create a JSBin account in order to participate, and will not have to follow all of the introductory activities if they are already familiar with HTML. Students will be able to work individually or in pairs, and can move through the activities at their own speed. This will hopefully serve to reduce the "boring" aspects of the activity. Additionally, some of the other coding studios had an open-ended activity at the end that allowed for students to continue working on the code past the class if wanted.

4.1.2 Coding Studios

Regardless of the placement of the HTML lesson, the other coding studios will also contribute to improving the curriculum. Each studio had their own challenges to solve in the redesign, in combination with the overall issues of the program.

Digital Literature

The Digital Literature lesson has been redesigned to reduce the amount of time spent lecturing the class and increase the challenges posed to the participants. Many students would become disengaged with the lesson when they turned away from the main screen in the classroom and would ask the same questions again and again. This was frustrating

for both the students and instructors. By reducing the amount of time spent lecturing, the goal is to remove this frustration and confusion. The key trade-off here is that with less lecturing, there needs to be another way to convey this information.

The iPython notebooks used in this class can be downloaded with examples already included in the cells rather than having the students type in each command one by one. This will allow for a quicker run-through of the different concepts without having to take students away from their computers or repeat instructions. Additionally, students will be encouraged to personalize the commands as they go through the tutorial, coming up with their own individual (possibly nonsensical) sentence by the end of the notebook.

It will be emphasized at this point that for much of computer science courses, you are either writing your own original code or modifying code that's already mostly written. The second portion of the lesson involves modifying the love letter code, and participants will be invited to change the lists of words in addition to the code. Once they complete the letter, they can either choose to write their own original text generator or continue to modify the code that's already in place. This change allows for more creativity and challenge, and allows students at different levels to come up with their own ideas rather than be tied to the pre-written love letter code.

In order to increase the enjoyment and interactivity of the coding studio, students will be invited to share the results of their code on Facebook, Twitter, Tumblr and other social media. In teaching the coding studio, instructors noticed that students would send pictures or screenshots to friends to show off their impressive and often amusing "love letters," "hate letters" or "business letters." Rather than remove the presence of social media and texting from the classroom, students can be motivated to challenge themselves to create a funny or weird result worth sharing.

Graphics

The Graphics lesson has been redesigned to include a more concrete "unplugged" portion, reduce the photo filter portion, and increase the challenges in the fish activity in order to motivate students. The unplugged activity has been fleshed out so that replication of the coding studio will be easier. Certain shapes were given to students in the original activity, making it a simple activity. Additionally, more students will be paired up for this unplugged activity, and the whole group will discuss the challenges faced by the partner giving direction versus the person drawing and following directions.

In the drawing activity, students will be encouraged to draw an example of what they're aiming to create, and then challenge themselves to replicate it as closely as possible. Again, this should be useful in increasing the motivation to continue with the activity, rather than settling for a simple solution.

The photo filters were difficult to get through without frustration, and so in the redesigned edition of the Graphics coding studio, students will be given the choice to pick whichever filter they are most interested in creating and modifying. This will hopefully increase interest in completing the activity.

Robotics

The robotics coding studio is largely left unchanged, but has been expanded to include an unplugged activity to open up the lesson. Students will be put into pairs and one student will be the "programmer" while the other is the "robot." The robot will be blindfolded and given instructions on how to get from one end of the room to the other, with obstacles in the way. The programmer will have to give clear and concise instructions to get their robot across the room.

The instructor will give them a limited set of instructions to use, such as "forward 2 steps" or "turn left 90 degrees" so as to simulate the type of instructions given to the

scribbler robots. This activity will hopefully help students get into the mindset of coding, and also create a fun environment from the start of the class.

4.2 Survey

The current version of the survey had a few problematic questions and made the subsequent analysis difficult. There were questions on the post-survey that weren't asked on the pre-survey, and therefore it was impossible to compare the before and after for these questions.

An additional statement added to the survey is "I consider computer science a part of the Liberal Arts." The reasoning behind adding this statement is to determine whether or not students see the value of including Computer Science in the Liberal arts curriculum. Seeing as they decided to attend a Liberal Arts school, it should be interesting to see if they consider computer science a liberal art. This question was added to both the pre-survey and the post-survey.

Another key question to include in order to make the best of the data is "Which coding studio did you attend?" Without this data, it was difficult to decipher where the problems were occurring causing students to report decreases in agreement to the statements concerning enjoyment and motivation in relation to computer science problems. In order to make this determination, it will help greatly to compare the experience of students based on the coding studio they attended. This question will only be added to the post-survey, as students may not have selected their coding studio before taking the pre-survey.

The last question added to the survey concerns the suitability of the coding activities as a part of the Language and Thinking program versus another part of the first year experience.

These coding activities should be a part of: (Circle your selection)
Language and Thinking Citizen Science Both Neither

One key concern in developing this curriculum was its applicability in the context of Language and Thinking versus the Citizen Science program, or if the program was suitable as a part of the first year experience at all. Students in fields like literature or art history may not see the applicability of computer science to their field, and an aspirational result of this program might be to see students embracing the applicability. In order to get that point across, the Language and Thinking program seems to fit best with that theme. However, Citizen Science and its readings might make more sense.

All of these additional questions will help to clarify the issues present in the curriculum, as well as highlight any aspects that work well. Keeping in the old questions will also help in allowing any future studies to be compared to the previous iteration of the program.

5

Conclusion

All in all, there were three steps in this project. The design and implementation steps of the program incorporated multiple different areas of study and engaged students at their own level of interest. The results and analysis stage revealed key effects of the curriculum on the students participating in the study, and highlighted any areas for improvement. The redesign addressed these issues and improved the curriculum for future use.

5.1 Final Discussion

The main takeaway from the survey results is that students gained an appreciation for the applications of computer science to other disciplines. There was a significant difference between the responses before and after the program indicating a positive change here. As more students agreed with the statement "Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business)," we can confidently conclude that the program had a meaningful effect on students opinion on the ability of computer science concepts to be helpful in other areas of study.

This is wonderful news, as this shows that students are aware of how taking computer science classes might assist them in their studies, regardless of the major they choose. However, the other question that had a real difference was "I enjoy solving computer science problems." Although this could be attributed to the fact that few students had taken computer science courses before the program, it still isn't a good sign. As a result, the redesign aims to remedy that by making the lessons more interactive and fun.

The anecdotal evidence and observations also indicated that students found the coding studios more engaging and fun than the HTML activity. Coupling this information with the fact that some professors had difficulty teaching the lesson on their own, asking for assistance from students and other professors, it made sense to focus energy on improving the in-class portion of the program.

The resulting lesson plans incorporated the HTML activity into the coding studios and enhanced the "unplugged" aspects of each lesson. With future use of the lessons and surveys, the hope is that the resulting data will show a positive increase in students agreement with the statement "I enjoy solving computer science problems," and other clear improvements.

5.2 Future Works

One problem that I ran into with this project was the inability to test the redesigned material. Despite several attempts, either zero, one or two students would show up to a non-mandatory coding workshop advertised through Facebook, posters and word of mouth. A second trial of the program would definitely help to validate improvements to the curriculum and to improve the program further.

An issue that came up with analyzing the survey data was the time it took to type up all of the data. In future applications of this study, online surveys could eliminate this process and lessen the time needed to analyze the data.

With more time and resources, it would have been interesting to see this study expanded to include surveys for Language and Thinking professors. It would be interesting to hear feedback about their experience, especially since students seemed to struggle with the HTML activity. Perhaps with this additional information, it would be easier to pinpoint exactly where the problems came up or what could be done to make this activity work better in the classroom.

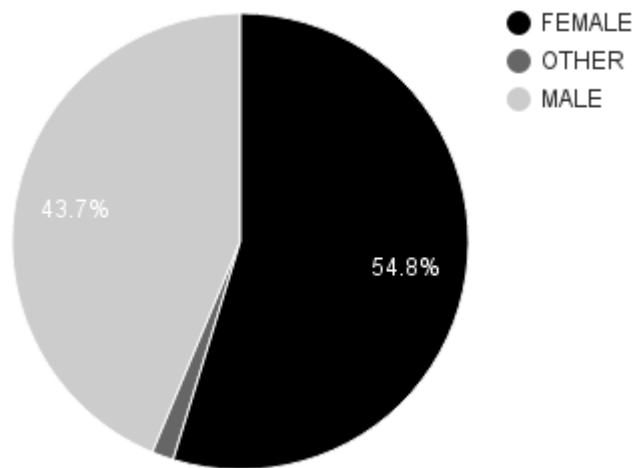
Another improvement could be polling students about their interest in options for the coding studios. If there was a large amount of interest in video games or computer vision, lessons could be modified to involve topics that are requested by students. Additionally, students might be interested in learning at different levels. For students that already had the opportunity to code in high school, they might not be challenged by the current lessons, and could benefit from more advanced coding studios. On the other hand, students who did not have access to coding might benefit from lessons that are less advanced.

Lastly, this program could benefit more than just first year students. As students near graduation, the benefits of coding skills become more apparent and immediate. Clearly, students in this program realized the applicability of computer science tools and techniques. If upperclassmen benefited from the same knowledge and coding ability, it could help them in their own area of study. Whether students decide to continue their education in computer science or apply the lessons to other interests, "computational thinking and problem solving skills" can improve students abilities to succeed in college and their career. [6]

Appendix

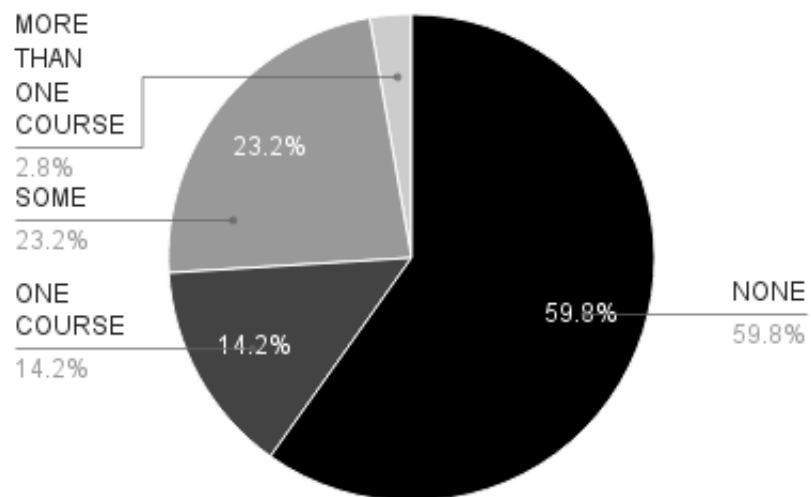
5.3 Survey Demographics

Gender of Participants



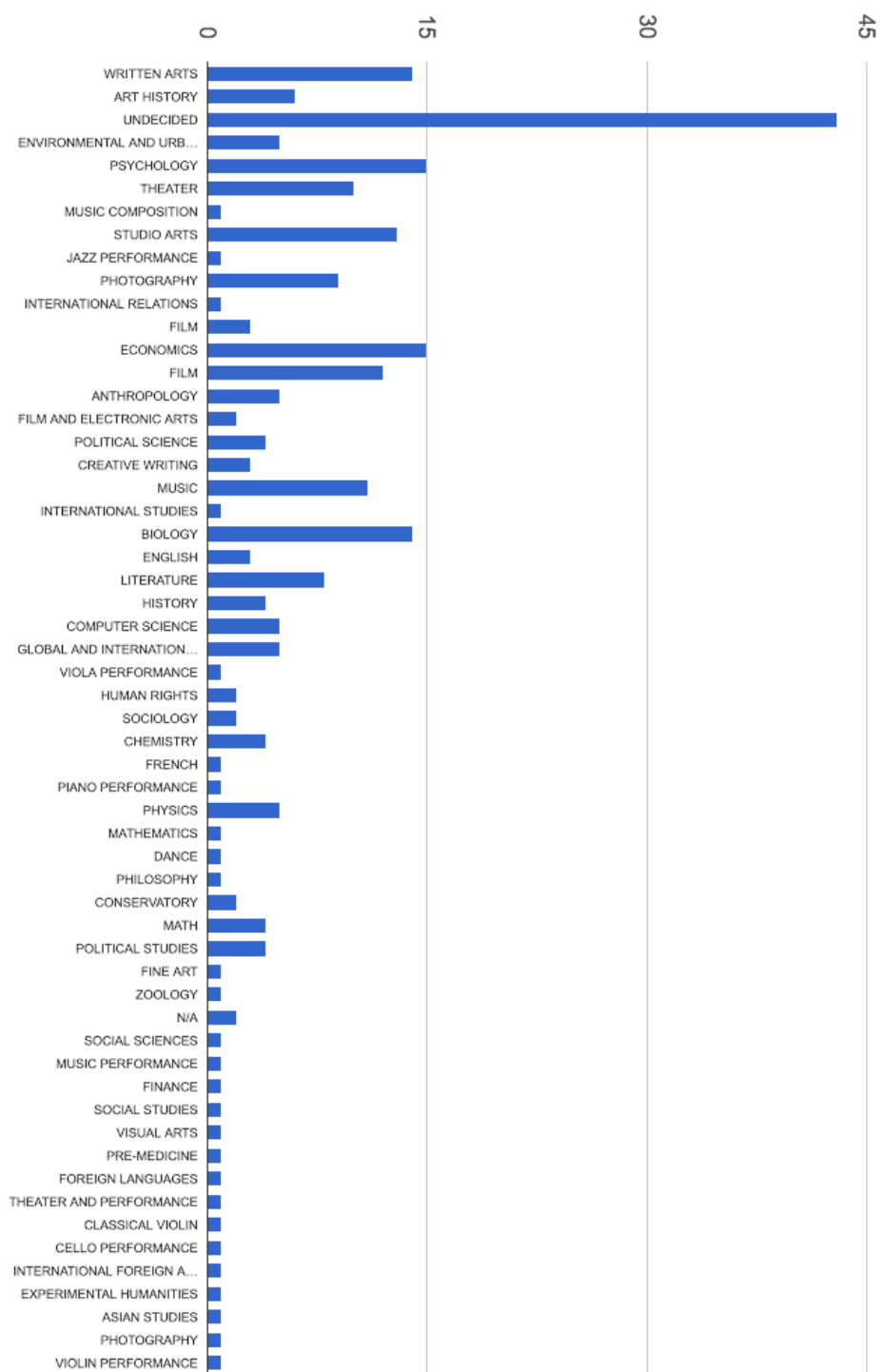
Gender	
Male Students	109
Female Students	149

Prior Experience



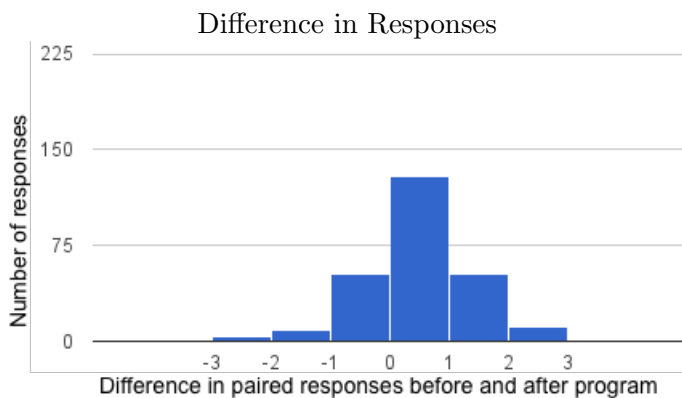
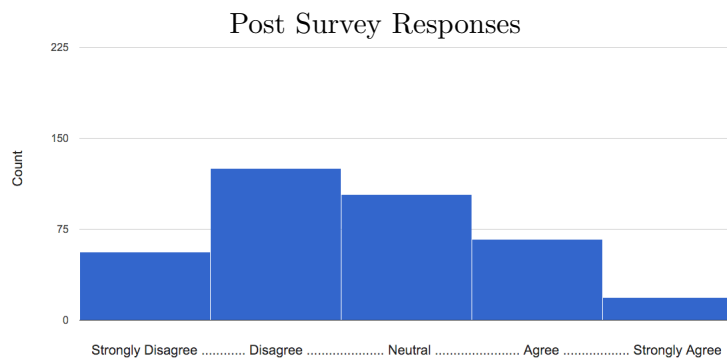
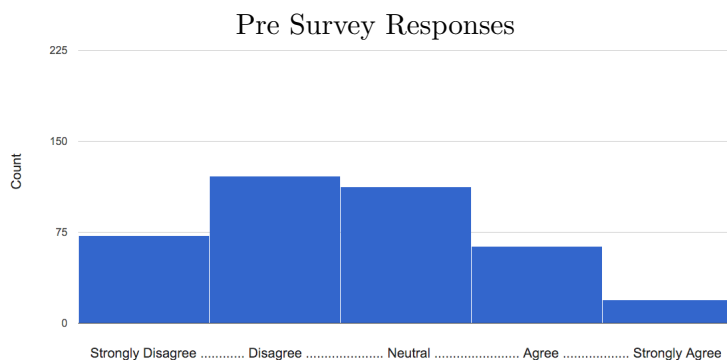
Prior Experience	
None	152
More Than One Course	7
One Course	36
Some	59

Student's Majors

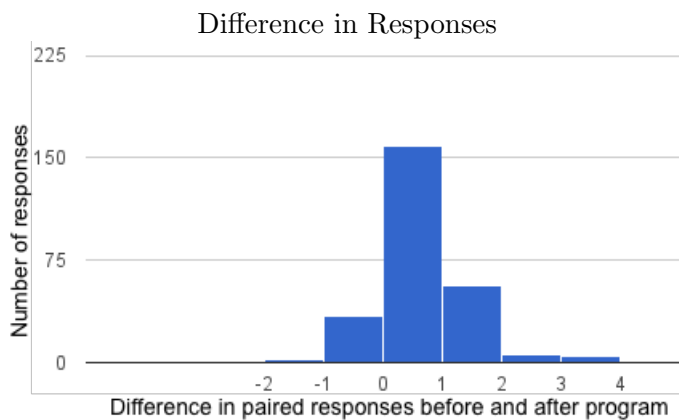
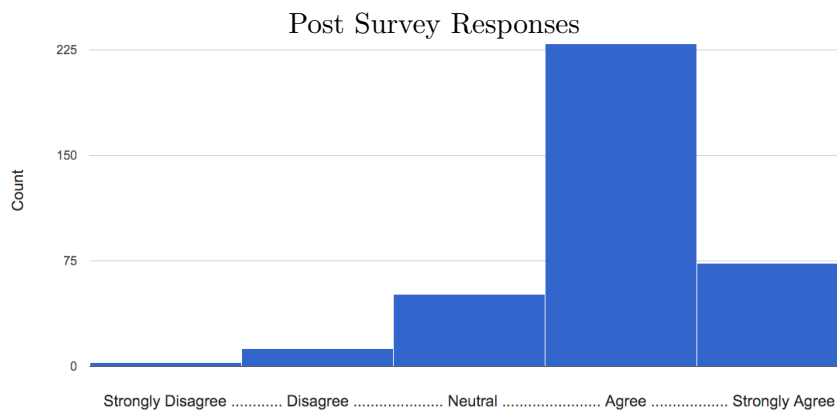
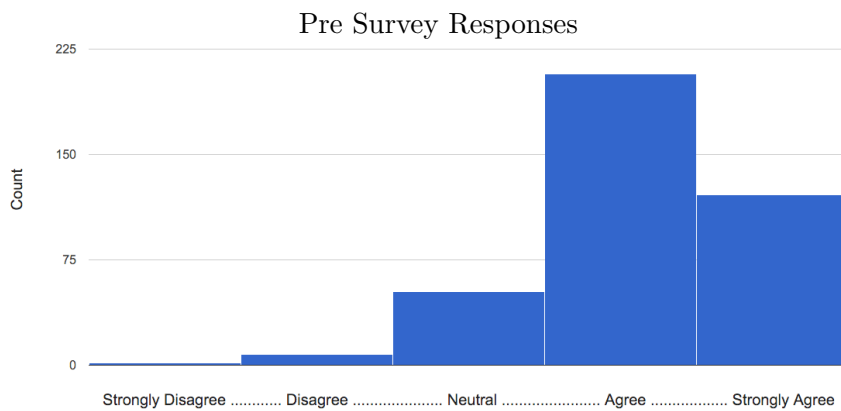


5.4 Survey Questions

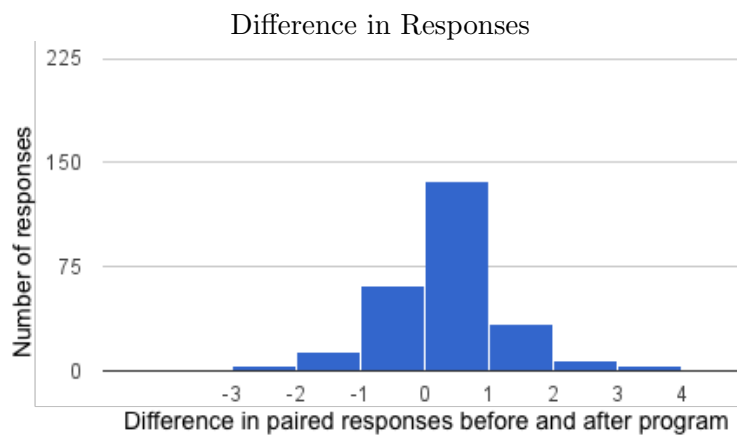
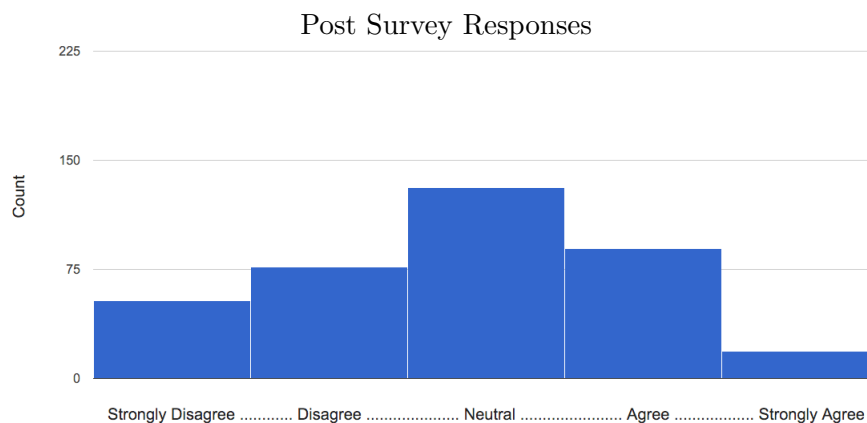
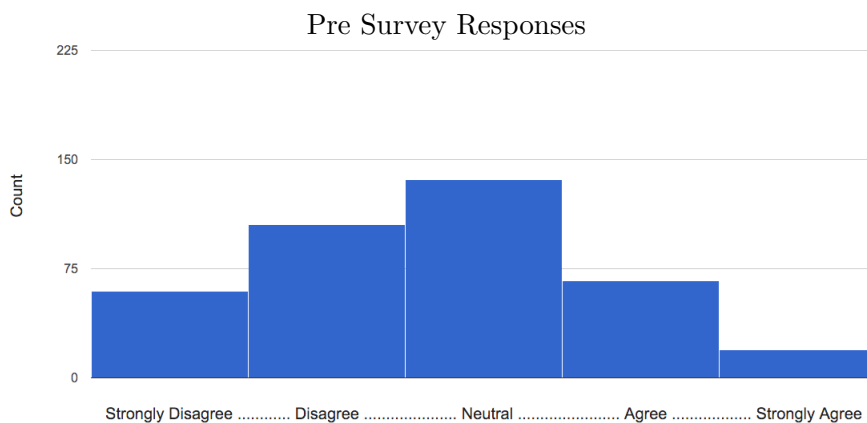
Q1: I think about the computer science I experience in everyday life.					
<i>t</i>	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-0.55132	0.5819	-0.13852868	0.07792262	-0.03030303	0.03393166



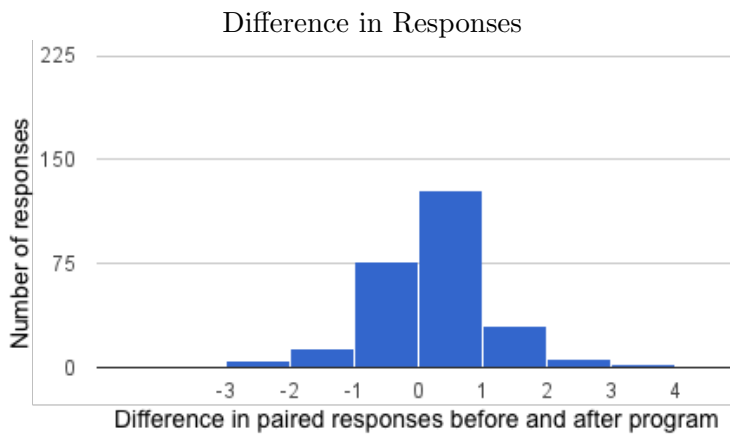
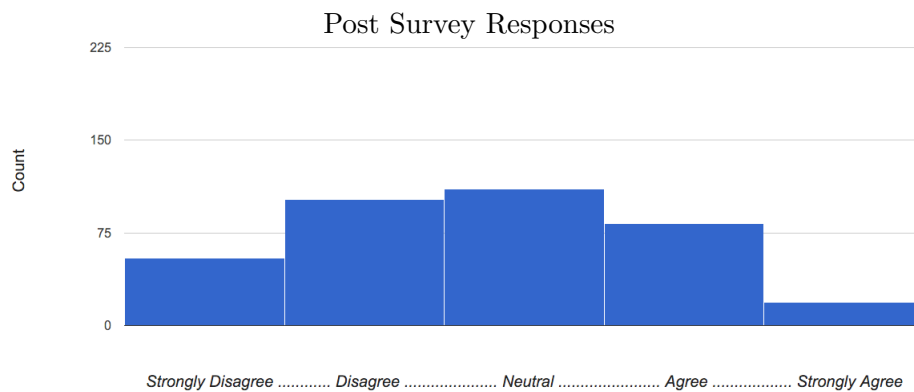
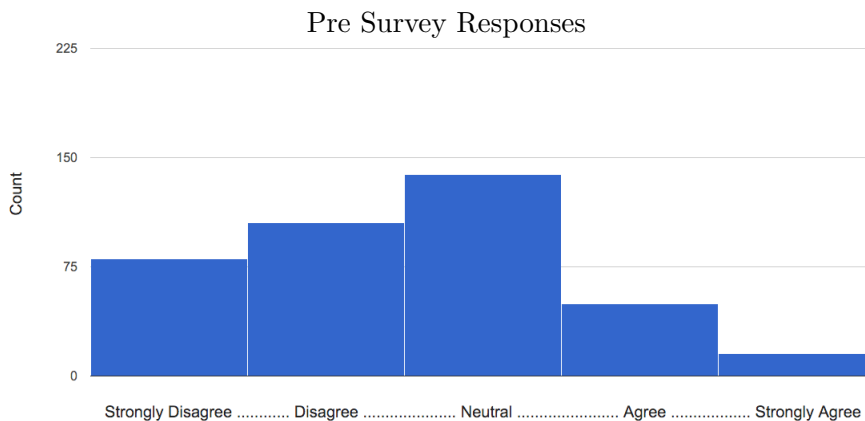
Q2: Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business)					
t	p-value	95% CI		Mean of the differences	Cohen's d
3.4497	0.0006535	0.06503319	0.237997128	0.1515152	0.2123146



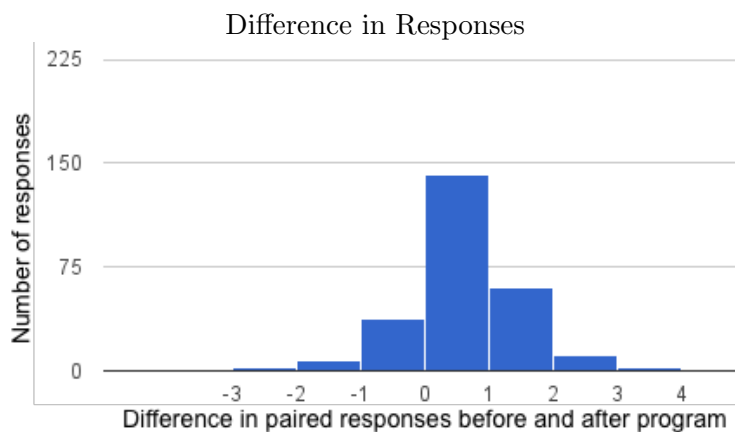
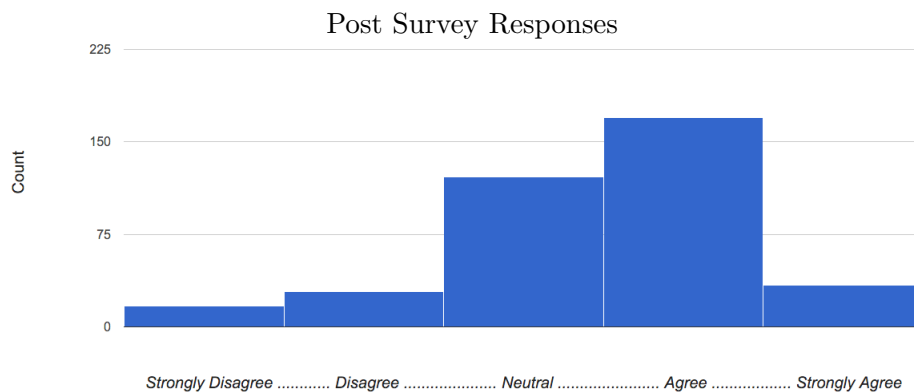
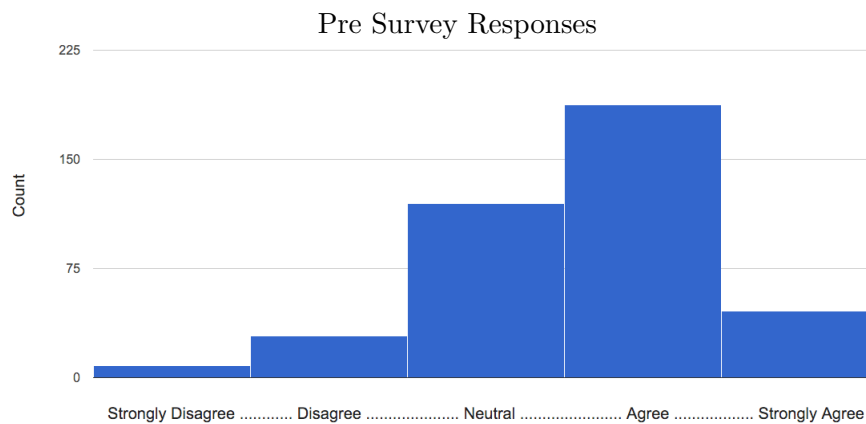
Q3: I find the challenge of solving computer science problems motivating.					
t	p-value	95% CI		Mean of the differences	Cohen's d
-2.9431	0.00354	-0.27817286	-0.05516047	-0.1666667	0.1811336



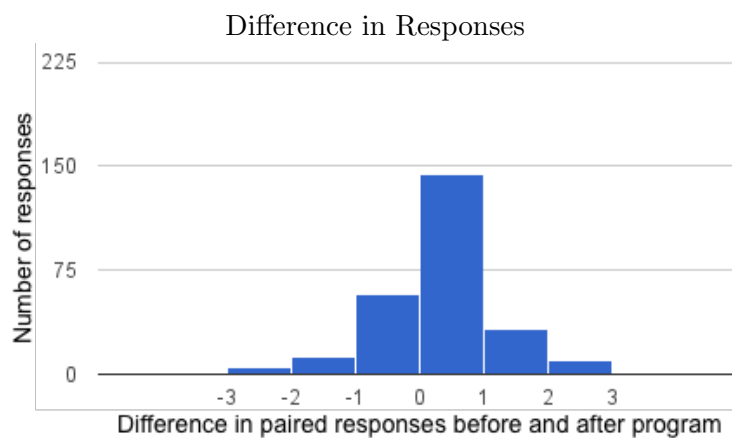
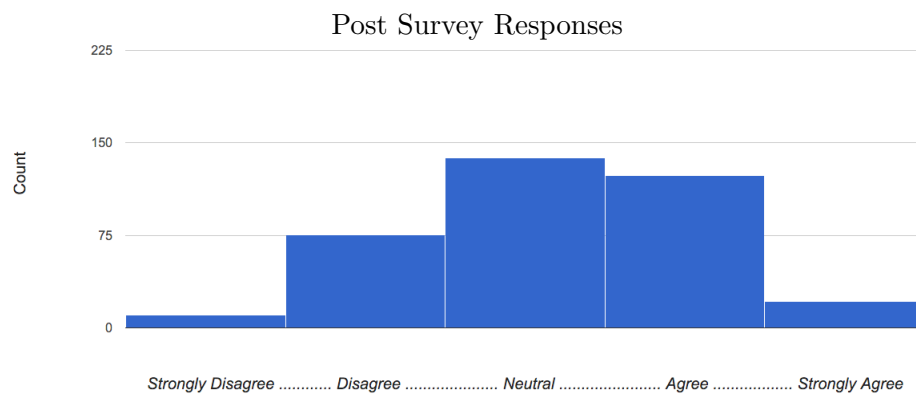
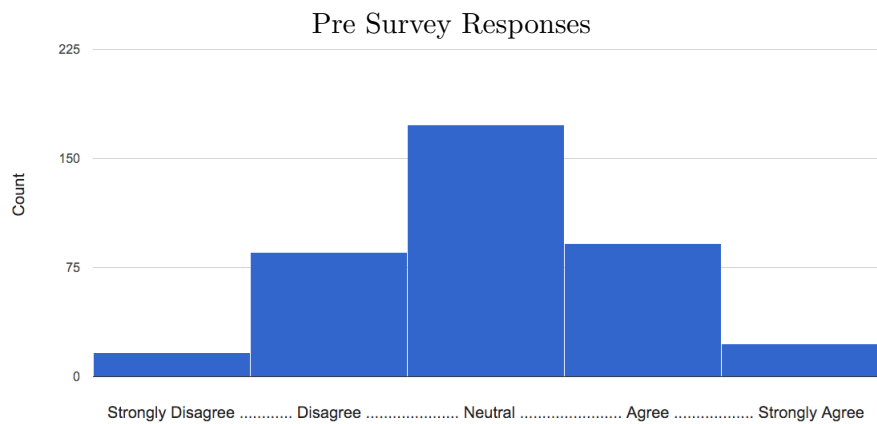
Q4: I enjoy solving computer science problems.					
t	p-value	95% CI		Mean of the differences	Cohen's d
-5.0665	7.64e-07	-0.3839788	-0.1690515	-0.2765152	0.311821



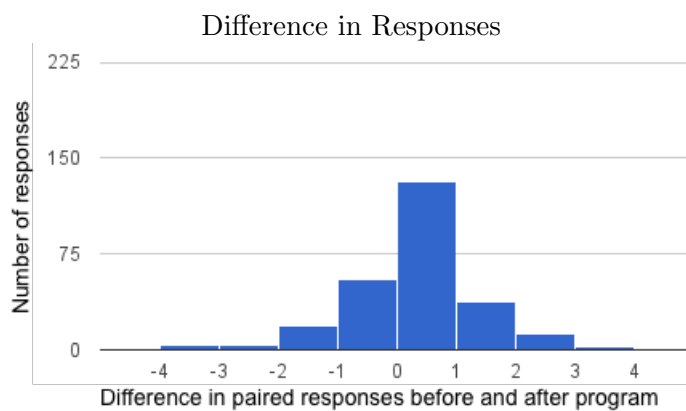
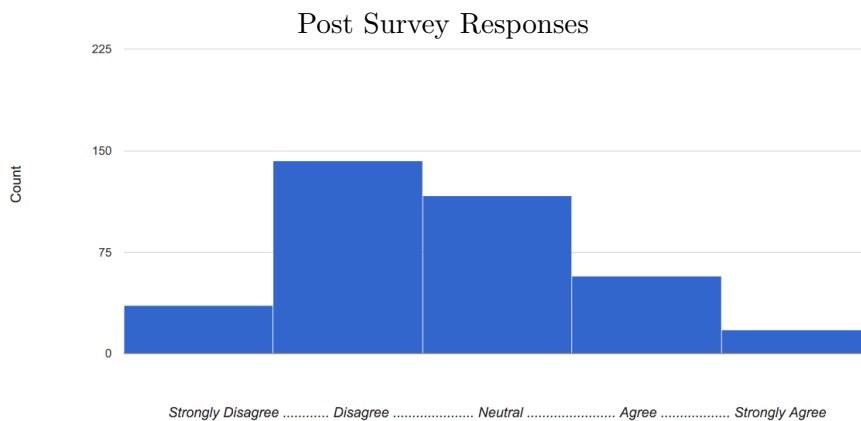
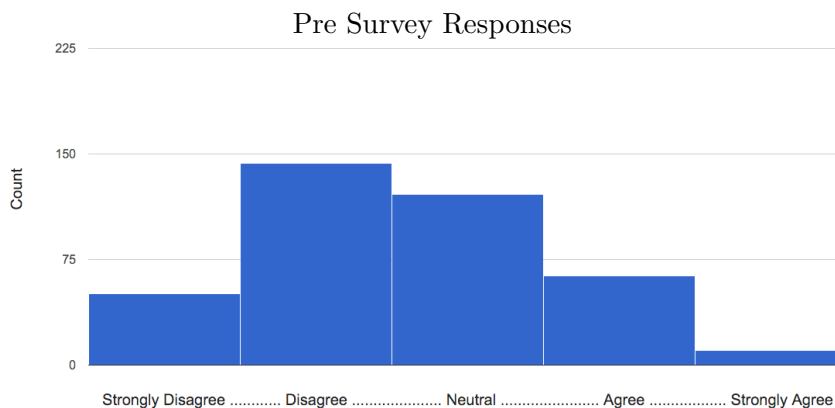
Q5: Reasoning skills used to understand computer science can be helpful to me in my everyday life.					
<i>t</i>	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
2.2976	0.02237	0.01679089	0.21805760	0.1174242	0.1414051



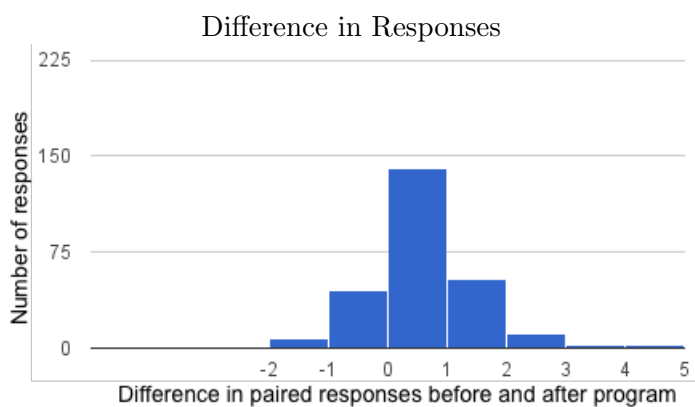
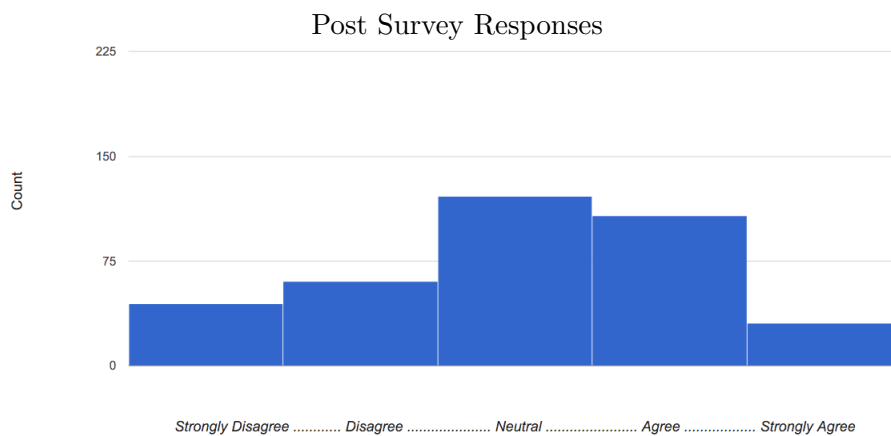
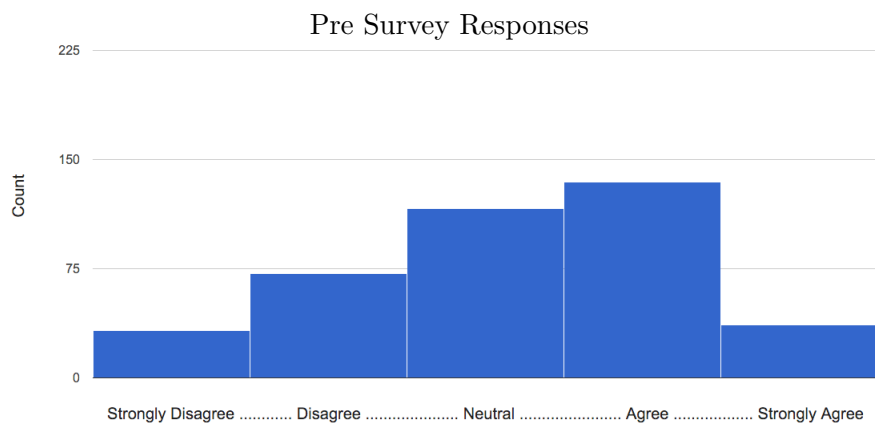
Q6: Learning computer science is just learning how to program in different languages.					
t	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-2.9903	0.003051	-0.27012970	-0.05562787	-0.1628788	0.1840402



Q7: The subject of computer science has little relation to what I experience in the real world.					
<i>t</i>	<i>p-value</i>	95% CI		Mean of the differences	<i>Cohen's d</i>
-2.5446	0.01151	-0.27547772	-0.03512834	-0.155303	0.1566088

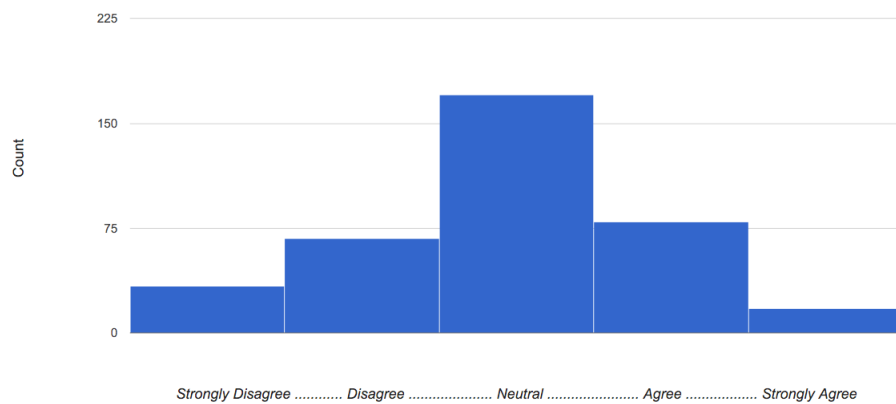


Q8: I am interested in learning more about computer science.					
t	p-value	95% CI		Mean of the differences	Cohen's d
1.9538	0.05178	-0.0007945371	0.2053399916	0.1022727	0.1202509

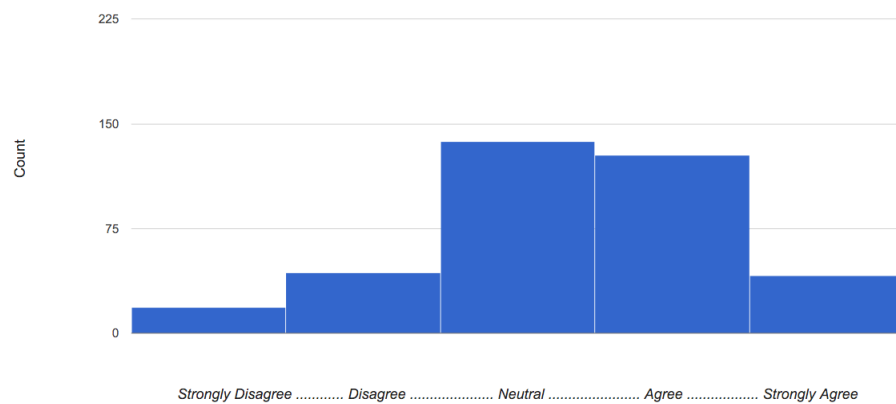


5.5 Additional Questions

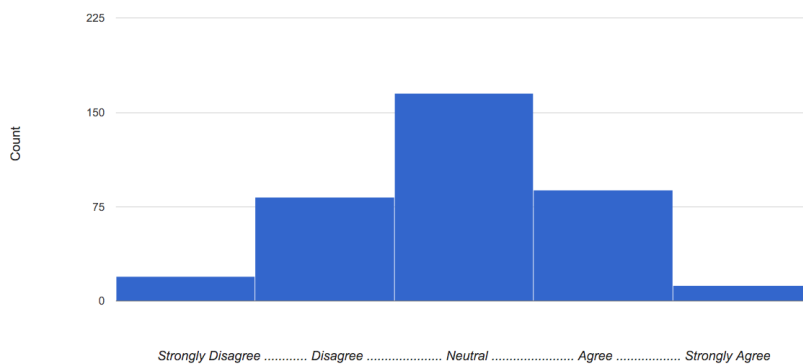
Q9: When I'm trying to learn something new in computer science, I find it useful to write a small program to see how it works.



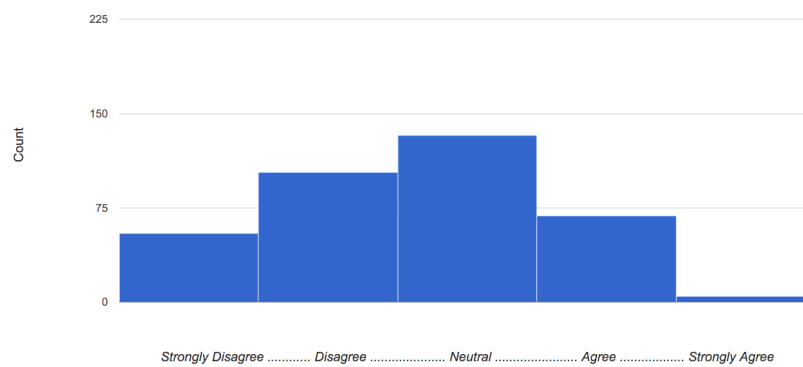
Q10: A significant problem in learning computer science is being able to memorize all the information I need to know.



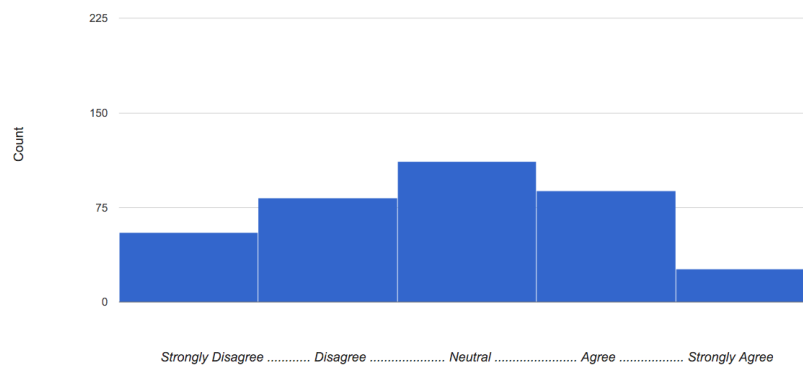
Q11: Understanding computer science basically means being able to recall something you've read or been shown.

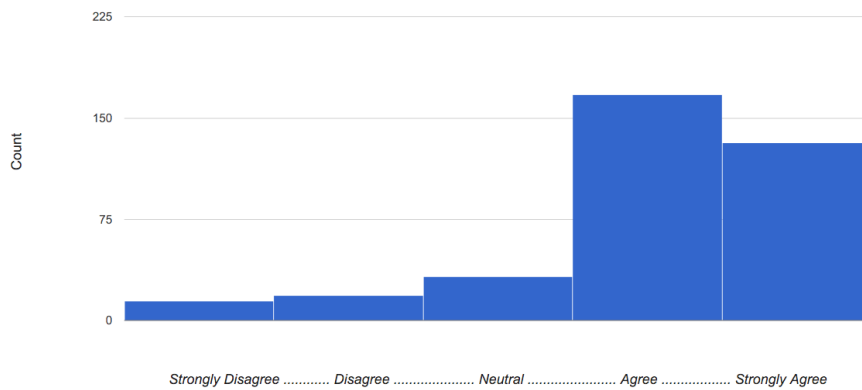
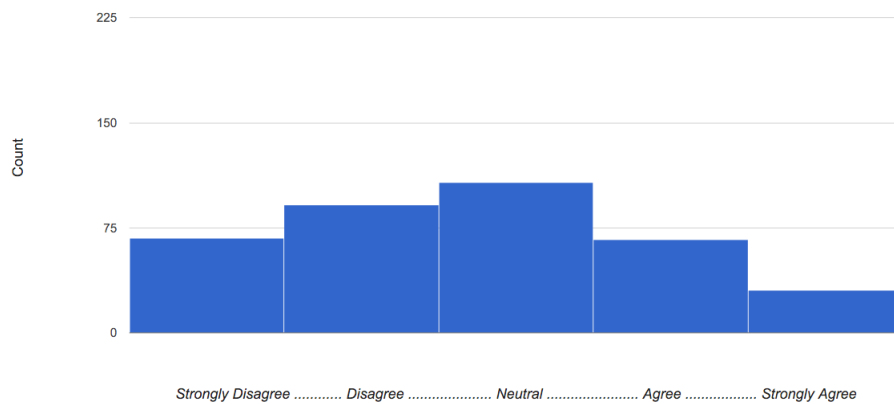


Q12: The readings about computing were relevant to L&T.



Q13: The in-class computing activities were illuminating.



Q14: I participated in the L&T coding studios.**Q15: I see myself writing another computer program someday.**

5.6 Surveys

5.6.1 *Old Surveys*

Language and Thinking IRB Proposal

What is the **title** of your project? Assessing Student Attitudes about Computing in Language and Thinking

Describe your **research question** briefly (approximately 250 words or less):

We are developing a set of materials about computing to be used during Language and Thinking (L&T) starting in August 2015. We plan to assess the effect of these materials – along with three hours of computing instruction – on attitudes toward and perceptions about computer science. Anonymous, paired pre-/post-surveys will be used to gauge student perceptions about computing. Many of the questions come from the established and validated [Computing Attitudes Survey](#)¹

Briefly describe **how** you will recruit participants. (e.g., Who will approach participants? What is the source of the participants?)

All L&T students that are 18 years and older will be asked to fill out the surveys in class during the first week (pre-survey) and last week (post-survey) of the three-week program.

NOTE: If you have supporting materials (recruitment posters, printed surveys, etc.) please email these documents separately as attachments to IRB@bard.edu. Name your attachments with your last name and a brief description (e.g., "WatsonConsentForm.doc").

The L&T teachers will distribute and collect the surveys.

Approximately how many individuals do you expect to participate in your study?

500 first-year Bard students.

Please describe any **risks and benefits** your research may have for your participants. (For example, one study's risks might include minor emotional discomfort and eye strain. The same study's benefits might include satisfaction from contributing to scientific knowledge and greater self-awareness.)

Time spent filling out the surveys will cut into an already tight schedule in L&T. If the students are already anxious about math and computing, reflecting on this during the survey could add to this anxiety. The benefits of participating would include improving future computing modules in L&T, improving other computing programs at Bard and beyond.

Have you prepared a **consent form** and emailed it as an attachment to IRB@bard.edu? Yes.

Please include here the **verbal description of the consent process** (how you will explain the consent form and the consent process to your participants):

The L&T teachers will read the consent form aloud and hand out the form. The teacher will assure every response is accompanied by a consent form. The surveys will have a unique ID that will be randomized according to section, and individual ID's assigned alphabetically.

What procedures will you use to ensure that the information your participants provide will remain **confidential**?

The surveys will not include any identifiable data (e.g. no name nor student #). And since the section number is randomized, student anonymity will be preserved.

¹ B. Dorn and A. E. Tew. Empirical Validation and Application of the Computing Attitudes Survey. *Computer Science Education*, 25(1), 2015 -- <http://dx.doi.org/10.1080/08993408.2015.1014142>

Consent Form

Project Title: Assessing Student Attitudes about Computing in Language and Thinking

Researchers: Sven Anderson and Keith O'Hara

Purpose:

Bard College has developed a set of materials about computing to be used during Language and Thinking. We hope to assess the effect of these materials, along with three hours of computing instruction, on attitudes and perceptions about computer science.

Procedures:

If you are 18 years of age or older, you will be asked to fill out a short one-page survey today and another short two-page survey at the end of the L&T program. Each survey should take no longer than fifteen minutes. Participation in the surveys is optional and will have no impact on your evaluation in L&T.

Risks/Discomforts:

Time spent filling out the surveys will cut into class time. If you are already anxious about math and computing, reflecting on this during the survey could add to this anxiety.

Benefits:

Your feedback will help improve future computing modules in L&T and other Bard programs. And when reported more broadly, lessons learned about computing in the context of *language and thinking* could inform computing education beyond Bard.

Compensation: There is no compensation for participation in this study.

Confidentiality of personal information:

The survey data collected about you will be kept anonymous. To protect your privacy, the surveys will be kept under a code number rather than by name. Your name and any other facts that might indicate your participation will not appear when the results of this study are presented or published.

In case of injury or harm:

If you are injured as a result of being in this study, please contact Sven Anderson, Principal Investigator (845)752-2322. Neither the Principal Investigator nor Bard College have made provision for payment or costs associated with any injury resulting from participation in this study.

Subject rights:

- Your participation in this study is completely voluntary.
- You have the right to change your mind and leave the study at any time without giving any reason, and without penalty.
- You will be given a copy of this consent form to keep.
- You do not waive any of your legal rights by signing this consent form.

Questions about the study or your rights as a research subject:

If you have any questions about the study, you may contact the Principal Investigators: Sven Anderson (845) 752-2322; sanderso@bard.edu or Keith O'Hara (845) 752-2359; kohara@bard.edu. If you have any questions about your rights as a research subject, you may contact the chair of Bard's Institutional Review Board, Pavlina Tcherneva (tchernev@bard.edu)

If you sign below, it means that you have read the information given in this consent form, and you would like to be a volunteer in this study.

Participant Name

Participant Signature**Date**

Signature of Person Obtaining Consent**Date**

Language and Thinking Pre-Survey

A. Intended Major _____

B. Gender _____

C. Prior computer programming experience (circle):

None

Some

One course

More than one course

Please rate the following statements:	strongly disagree	disagree	neutral	agree	strongly agree
1. I think about the computer science I experience in everyday life.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2. Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business).	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3. I find the challenge of solving computer science problems motivating.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4. I enjoy solving computer science problems.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
5. Reasoning skills used to understand computer science can be helpful to me in my everyday life.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6. Learning computer science is just learning how to program in different languages.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7. The subject of computer science has little relation to what I experience in the real world.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
8. I am interested in learning more about computer science.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Language and Thinking Post-Survey

A. Intended Major _____

B. Gender _____

C. Prior computer programming experience (circle): None Some Good deal

Please rate the following statements:	strongly disagree	disagree	neutral	agree	strongly agree
1. I think about the computer science I experience in everyday life.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2. Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business).	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3. I find the challenge of solving computer science problems motivating.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4. I enjoy solving computer science problems.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
5. Reasoning skills used to understand computer science can be helpful to me in my everyday life.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6. Learning computer science is just learning how to program in different languages.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7. When I'm trying to learn something new in computer science, I find it useful to write a small program to see how it works.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
8. A significant problem in learning computer science is being able to memorize all the information I need to know.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
9. Understanding computer science basically means being able to recall something you've read or been shown.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
10. The subject of computer science has little relation to what I experience in the real world.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

11. I am interested in learning more about computer science.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
12. The readings about computing were relevant to L&T.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
13. The in-class computing activities were illuminating.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
14. I participated in the L&T coding studios.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
15. I see myself writing another computer program someday.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5.6.2 New Surveys

Pre-Survey

A. Intended Major _____

B. Gender _____

C. Prior computer programming experience (circle):

None

Some

One course

More than one course

D. Coding Studio attended during L&T (circle):

Digital Literature

Computer Graphics

Programming with Robots

I did not attend

Please rate the following statements:	strongly disagree	disagree	neutral	agree	strongly agree
1. I think about the computer science I experience in everyday life.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
2. Tools and techniques from computer science can be useful in the study of other disciplines (e.g., biology, art, business).	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
3. I find the challenge of solving computer science problems motivating.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
4. I enjoy solving computer science problems.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
5. Reasoning skills used to understand computer science can be helpful to me in my everyday life.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6. Learning computer science is just learning how to program in different languages.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7. The subject of computer science has little relation to what I experience in the real world.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
8. I am interested in learning more about computer science.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
9. I consider Computer Science a part of the Liberal Arts.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5.7 Lesson Plans

Lesson Plan

HTML Workshop

Objective

- Want students to learn some HTML that will help them to create their own websites.
- Also want students to understand how the PageRank algorithm works.
- Students should understand that coding is more than just memorization.
- Encourage students to come up with their own idea of how to figure out the most powerful website based off of who they link to.

Time Allotment: 50 minutes (with time before and after for surveys)

Implementation

Learning Context

- Some students will already have taken the HTML lesson.
- Ask students how they think Google selects the best sites to use in search results.

Procedure

- a. Engage students in the material - ask them how they think Google ranks their searches, why HTML is important, and what coding experience they have so far.
- b. Show students an example of how to look at the code that's used to build websites - an example could be the Space Jam Website.
- c. Guide Students through the activities on the jsbin website. They can select one random one, and then try to figure out the code surgery activity.
- d. After each activity, especially code surgery, ask the students to look at each other's solution, and encourage them to challenge themselves.
- e. Ask them to create their own website - maybe even using material from the previous activities.
- f. Guide students through the PageRank activity - have the tokens set up ahead of time, and ask students to write both their name and their link on the page.

Materials and Preparation

A. Materials:

- a. Wooden nickels
- b. Paper
- c. A bag to mix up the names in
- d. Whiteboard marker
- e. Projector
- f. Surveys
- g. Pencils

B. Preparation:

- a. Split wooden nickels into sets of 12.
- b. Write URL(s) on the board.
- c. Set up projector.
- d. Cut up paper into pieces for students
- e. Print out Surveys

Digital Literature Lesson Plan

A. Crash Course in Programming

1. explain function: take inputs and give outputs
 - a. defining functions like defining words
 - b. `def greeting1():`
`print "Roses are red!"`
2. explain variable/parameter
 - a. `def greeting(color):`
`print "Roses are" + color`
`def sentence(adjective, pluralnoun)`
`print (adjective + " " + pluralnoun + " are " + adjective)`
3. `---- from random import * ----`
 - a. `randint` --- explain in its own cell
4. Randomness
 - a. `def greeting2()`
`rnum = randint(1,3)`
`if rnum == 1:`
`a;lskdjf`
`elif rnum == 2:`
`asldkfj;`
`else:`
`a;lskdjf`
5. Lists and choice -- similar to what's on the board!
 - a. `nouns = ["BOX", "PERSON", "SHOELACE"]`
 - b. `def sentence():`
`print("The " + choice(nouns) + " walked down the road.")`

B. Strachey's Love Letter

1. Walk them through the love letter code, going through the steps we introduced earlier.
 - a. Ask the students to explain what's going on!
 - b. If they get stuck, give them hints!
 - c. Can always go back to the previous activity to relearn any key aspects
2. Encourage them to change the code!
 - a. Start with adding more to the lists!
 - b. Remind them to run each cell in order!
 - c. Then change sentence structure!
 - d. Walk around and help them!

Graphics Studio Lesson Plan

Plan to demonstrate all basic functions on projector and have students try them. Working in pairs is optional.

- Explain window coordinate system.
- Introduce **setup()** and **main()**, explain how **setup()** is run once at the start and **main()** loops continuously.
 - Show **size()**
 - **width, height**
- Geometry Basics:
 - **ellipse()**
 - **rect()**
 - **triangle()**
 - **stroke(), noStroke()**
- Color Basics:
 - **background()** //explain how to reset **background()** each draw cycle.
 - **fill()**
- Input Basics:
 - **mouseX, mouseY**
- **First Activity: Custom Fish**
 - Given skeleton-code of basic fish, modify with own geometry and color stuff
 - May also modify **swim()** function if they want, change fish's behavior
- More input: **keyPressed, keyCode**
- Show how to get webcam input working
 - **import processing.video.***
 - **image(cam,0,0)**
 - **cam = new Capture(this, 640, 480)** //maybe don't explain this stuff, just provide skeleton code that does it? Focus instead on pixel looping in context of webcam input.
- Explain for loops, pixel looping, getting/changing/processing pixels, **color()**
- If time, explain **Pimage** and image importation?
- **Second Activity: 3 Choices**
 - Choice between 3 possible activities. Potentially put to vote?
 - **Paddleball:**
 - Modify skeleton paddleball game to add collision, points, etc.

- **Webcam:**
 - Play around with making image filters (based on keyPressed) with pixel looping
- **Drawing:**
 - Make a drawing program that changes the “brush” (on keyPressed)
- **Aspirational Code**
 - Recursive Tree, Recursive Circles
 - Flocking Example

Instructor Guide -- Do Not Hand out to Students!

Getting to know your robot

Introduction:

1. Explain a bit about robotics and the motivations behind it.(3 or 4 D's of robotics) Talk about the advantages to automating processes too.
2. Demonstrate on the projector an example of using Calico. Emphasis the shell's purpose and why we would use scripts/programs opposed to writing in the shell.
3. Explain that we're using Python, a programming language, as well as a module called Myro to communicate with the scribbler robot.
4. Walk them through the first section.

The programming language in use for these activities is called python. In Python, and most programming languages in order to get things done we use things called functions. Functions can take inputs and return nothing, some could take no inputs and return information, and some functions take inputs and return some output. These inputs are called parameters.

Let's begin by entering the command `setName("_____")` in the shell (the left side of Calico).

Where the blank is, you should insert whatever name you'd like to give your scribbler robot.

For example, you could type in:

```
setName("Robot123")
```

After we give our robot its new name we can retrieve this data with the function `getName()` to see our robot's name.

Self Portrait

Use the robot's camera to take your picture and save it to a file using the following code snippet:

```
p = takePicture()  
show(p)  
savePicture(p, "me.jpg")
```

Instructors: This will be saved in the calico directory. Let them know where to find their image, and when each studio is finished, delete the old images, for privacy reasons.

Additionally, this is a good place to remind students of the syntax of calling a function.

Driving Your Robot

Instructors: Remind students of the `stop()` function! Write it on the board!
Tell students to put their robot on the ground!

There a variety of ways of getting your robot to move; in this lab we will stick to using motors. The function `motors` asks for two power values in the range $[-1, 1]$ for both the left and right wheels. Experiment with the `motors()` function. Complete the following table describing the robot's behavior for the uses of `motors`:

<code>motors(1, 1)</code>	
<code>motors(0, 0)</code>	
<code>motors(-0.8,-0.8)</code>	
<code>motors(0, -1)</code>	
<code>motors(1, -1)</code>	
<code>motors(0, 0.75)</code>	
<code>motors(1, 0)</code>	
	turns in a counterclockwise arc

Scribbling

Instructors: Reiterate the difference between the shell and writing your own program. Want to show them an example of a program here, as well as how to define your own function on the projector screen. Maybe even use the whiteboard to draw out the syntax, so they can use it for reference.

5. Unplugged Activity: Have students give instructions to another blindfolded student, as if they were a robot.
 - a. Should stress the idea that these are similar to functions in computing
 - b. This also shows how important it is to be clear and concise when communicating with computers.

Use your robot to draw a square, circle, rectangle, or any other shapes using
(a) the `gamepad()`
(b) the functions you learned.

Link to student copy:

<https://docs.google.com/document/d/17pIRsuvROVInvezhzaow99YVUwIkCsffMqheO2zpRoU/edit?usp=sharing>

Bibliography

- [1] *About Project Jupyter*, available at <http://jupyter.org/about.html>.
- [2] *AP Program Participation and Performance Data 2015* (2015), available at <http://research.collegeboard.org/programs/ap/data/participation/ap-2015>.
- [3] Ben Wood, Howard Aiken, and Howard P. Brooks Jr., *The Origins of Computer Science*, available at <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/compsci/>.
- [4] Brian Dorn and Allison Elliott Tew, *The Computing Attitudes Survey* (2015, January), available at <http://faculty.ist.unomaha.edu/bdorn/cas.html>.
- [5] *CS Education Statistics* (2016), available at <http://www.exploringcs.org/resources/cs-statistics>.
- [6] *CSNYC* (2015), available at <http://www.csny.org/csny>.
- [7] *CS Unplugged: Activities* (2016), available at <http://csunplugged.org/activities/>.
- [8] *Curriculum* (accessed 2016), available at <http://www.allstarcode.org/curriculum/>.
- [9] *FACT SHEET: President Obama Announces Computer Science For All Initiative* (2016, January 30), available at <https://www.whitehouse.gov/the-press-office/2016/01/30/fact-sheet-president-obama-announces-computer-science-all-initiative-0>.
- [10] *Features*, available at <https://JSBin.com/help/features>.
- [11] *Frequently Asked Questions*, available at <https://trinket.io/faq>.
- [12] *Girls Who Code: About* (2016), available at <http://girlswhocode.com/about-us/>.
- [13] John R. Rice and Saul Rosen, *History of the Department of Computer Sciences at Purdue University*, available at <https://www.cs.purdue.edu/history/history.html>.

- [14] Jorge Luis Borges, Donald A. Yates, James East Irby, and Andr Maurois, *Labyrinths: selected stories & other writings*, New Directions Pub. Corp., New York, 1964.
- [15] Luigi Federico Menabrea and Ada King Lovelace, *Sketch of the Analytical Engine Invented by Charles Babbage, Esq.*, Taylor and Francis, London, 1843.
- [16] Massimo Franceschet, *PageRank: Standing on the shoulders of giants* (2010, August 14).
- [17] Nick Montfort and Natalia Fedorova, *Small-Scale Systems and Computational Creativity* (2012).
- [18] *Promote Computer Science* (2016), available at <https://code.org/promote>.
- [19] Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, 1980.
- [20] Vannevar Bush, *As We May Think* (1945, July), available at <http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/>.
- [21] William M.K. Trochim, *Likert Scaling* (2006, October 20), available at <http://www.socialresearchmethods.net/kb/scallik.php>.