COSI 2A - Introduction to Real Life Programming Using Python

Designing and Evaluating an Active Learning style
Introduction to Programming Course for
Non-Computer Science Majors
With a Focus on Real Life Applications

A Thesis Presented in Partial Fulfillment of
the Honors Bachelor's Degree

Arya Boudaie

Abstract

This senior thesis will describe the process of designing a CS course for non-majors, with a focus on real life applications of programming. We will also discuss the implementation of the course, and the results of the implementation, as a proposal for research into this type of course. To implement this idea, we created a two-unit course in Python, with the first unit on core Python skills, and the next unit on real life applications of those skills. My research question is as follows: Can we make an introduction to programming class in an interdisciplinary setting that is accessible to beginners and non COSI majors, as well as gives students more practical skills, while still learning the fundamentals? Can this be used as a replacement for intro-CS for all students?

Department of Computer Science
Under the supervision of Dr. Tim Hickey and Dr. Antonella Di Lillo
Brandeis University
May 2017

Contents

Abstract

Acknowledgments

1 Introduction **1**
- 1.1 Background for Project. . . . 1
  - 1.1.1 COSI 11A - Introduction . . . 1
  - 1.1.2 Problems with COSI 11A . . . 2
  - 1.1.3 Background of COSI 2A . . . 4
- 1.2 Goals and Research Questions . . . 5

2 Demographics **5**
- 2.1 Recruitment . . . 5
- 2.2 Demographics of Students . . . 6

3 Overview of Course **10**
- 3.1 Unit 1 - Rapidly learn core Python . . . 10
- 3.2 Unit 2 - Solving Real World Problems . . . 11
  - 3.2.1 Goals . . . 11
  - 3.2.2 Programming Assignment . . . 12
  - 3.2.3 Jupyter Notebooks . . . 13
- 3.3 Methods on Objects . . . 14
  - 3.3.1 Text and CSV files . . . 15
  - 3.3.2 Flask . . . 16
  - 3.3.3 APIs . . . 18
  - 3.3.4 Final Project . . . 19
- 3.4 Final Goal . . . 21

4 Results **21**
- 4.1 Successes . . . 24
- 4.2 Things to improve for future implementations . . . 25
  - 4.2.1 Mentorship . . . 25
  - 4.2.2 Atmosphere of Course . . . 27
  - 4.2.3 Size of Course . . . 27
  - 4.2.4 Rigor and Pace of Course . . . 28
  - 4.2.5 Reliance on Spinoza . . . 29
  - 4.2.6 Final Project, Groups, and POGIL . . . 29
- 4.3 Future Steps . . . 30

5 Related Work **31**
- 5.1 Computer Science Courses for Non Majors . . . 31
- 5.2 POGIL and Jupyter Notebooks . . . 33

6 Conclusion **33**

7 Contributions **33**

References **35**

Appendices **36**

A Jupyter Notebook For Strings/Methods **36**
A.1 Part 1: Review . . . 36
A.1.1 Project 1: . . . 37
A.2 Part 2: Built in Methods . . . 37
A.2.1 How to find methods! . . . 38
A.2.2 AFTER YOU HAVE TRIED MESSING WITH CODE . . . 38
A.3 Project 2: . . . 39
A.4 Another way to learn about methods . . . 39
A.5 Project 3 . . . 39

B Jupyter Notebook on Reading and Writing Files in Python **39**
B.1 Introduction . . . 40
B.2 Files1 . . . 41
B.3 Digital Humanities . . . 41
B.4 Line Numbers! . . . 41
B.4.1 Files 2: Modify the code in the previous example to also print the line before and the line after each occurrence of the word "dog". . . . 42
B.5 How to store information in files. . . . 43

C Jupyter Notebook for How to Use and Manipulate CSV Files in Python **43**
C.1 Part 1 - What is a CSV file . . . 43
C.2 Part 2 - Reading CSV Files . . . 44
C.2.1 Part 2a - Reading them manually . . . 44
C.2.2 Part 2b - Using the CSV library . . . 45
C.3 CSV1 . . . 46
C.4 CSV2 . . . 47
C.5 Part 3: Writing a CSV file . . . 48
C.6 Real Data . . . 48
C.6.1 Real Estate data . . . 48
C.6.2 Number 2: . . . 50

D Jupyter Notebook for Introduction to APIs **50**
D.1 Part 1: What is an API . . . 50
D.2 Part 2: How to use a Python-ported API . . . 50
D.2.1 Part 2A: Indico API . . . 50
D.3 Exercises . . . 54
D.3.1 Question 1 . . . 54
D.3.2 Question 2 . . . 54

List of Figures

1 Enrollment in COSI courses . . . 1
2 Number of Computer Science Graduates at Brandeis[1] . . . 2
3 Hello World program in Java . . . 3
4 Example usage of REPL to find middle three elements of a list in Python . . . 4
5 Invitation to Biology Department . . . 6
6 Majors represented in the course (35 unique) . . . 9
7 Example of a Spinoza problem (return the first vowel), with unit tests done on an almost correct solution . . . 10
8 Sample program for PA1 . . . 13
9 Hello World program in Flask, as well as another page that takes an argument . . . 16
10 Function to render factors HTML . . . 17
11 HTML for Factors Page . . . 17
12 Rendered Factors Website . . . 18
13 Sample Indico library code. Sentiment is 0-1, where 1 is positive, and 0 is negative. . . . 19
14 Graph of student progress on PA1 (Old v New) . . . 25
15 Calender of weekly office hours . . . 26
16 Example Biography of TA . . . 26
17 Sample Spreadsheet to convert to CSV . . . 44

List of Tables

1 Most popular courses at Brandeis Spring 2017 (by single section) . . . 7
2 Most popular courses at Brandeis Spring 2017 (adding sections together) . . . 7
3 Gender Breakdown . . . 7
4 Race Breakdown . . . 8
5 Math Background of Students . . . 8
6 Years in School . . . 8
7 Programming Background of Students . . . 8
8 First Generation College Students . . . 8
9 International Students . . . 8
10 Found out about course through . . . 9
11 Departments Students heard about course from . . . 9
12 Considering CS Major/Minor . . . 9
13 Yes/No Questions about effect of course . . . 21
14 Difficulty of Course . . . 22
15 Ranking 1-5 the effect of the course and materials used . . . 22
16 Students Ranking Skills 1-5 . . . 22
17 Students Ranking the Course Goals 1-5 . . . 23
18 Comparing Course to COSI 11A, sorted by Δ . . . 23

Acknowledgments

I would like to thank the following people:

- Professor Tim Hickey, for spending countless hours in meetings with me to make COSI 2A as good as we did, as well as advising me through this thesis, and giving me my first job as a teaching assistant for COSI 11A sophomore year, and generally just supporting the ambitions of the computer science students at Brandeis.
- Professor Antonella Di Lillo, for agreeing to be the second reader on this paper, as well as being my first professor in the department, and trusting me to be the head TA for COSI 11A, showing me the fun of teaching others computer science, as well as the fun of creating my own assignments for people to finish. In general, you have always encouraged me to be my best and stuck up for me when it felt like nobody else was, and the dedication you have to the CS department at Brandeis is remarkable.
- Professor Jim Storer, for being my advisor for the computer science major from day one, and always encouraging me to do the crazy things I've done, such as take Theory of Computation my first year, or go to India for a study abroad in graduate level computer science.
- Michel Paul, my high school "MACH"[2] teacher, for opening the world of programming and computer science education to me, and fighting the high school administration to make the course happen. Six years after taking your class, I'm still amazed at what I've learned. My success at Brandeis would not be possible without your initial guidance and steering, and I'm infinitely grateful for your sacrifice to the school system that didn't realize what the importance of your class was.
- The Teaching Assistants for COSI 2A: Christa Caggiano, Talie Massachi, Kiana Khozein, Claire Sun, Joseph Tinianow, Monica Shi, Rebecca Holman, and Vanio Dos Santos, for being amazing friends and executioners of this crazy plan Tim and I had.
- Christa Caggiano (again), for always encouraging me to always be my best, encouraging me to write a senior thesis, helping me make awesome graphs, and sticking with me through the toughest times.
- Fatima Abu Deeb, for creating Spinoza to help make this class a success, and for helping me answer TeachBack questions all semester.
- The entirety of the COSI 2A class (of which there are too many people to name), for always putting their best foot forward and working with us to make the best possible class this semester.
- Sofiya Semenova, Christine Kim, and Kiana Khozein (again), for being my first friends in the computer science department, working together to make BITMAP the best club it could be, and always pushing me to excel in what I'm doing.
- Ryan Marcus, Solomon Garber, and Sofiya Semenova (again), for thoughtful conversations about CS education we've had over the years that have shaped my views and ideas for this course.
- My mother and father, for giving me the opportunity to come study computer science so far away, and always rooting for me no matter where I am or what I'm doing.
- My brother (Eli Boudaie) and sister (Celine Boudaie) for being the best siblings I could ever dream of, and making me proud all the time.

[2] Math Analysis Computational - teaching Pre-Calculus and other important lessons through Python programming

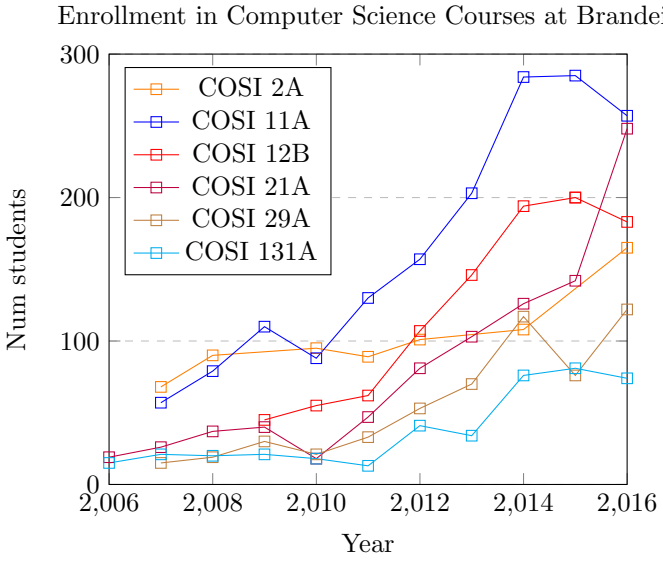
1 Introduction

1.1 Background for Project.

1.1.1 COSI 11A - Introduction

At universities all over the country, but specifically at Brandeis University, the enrollment in computer science courses has grown at an unprecedented rate. In the 2016/2017 year, the Introduction to Programming course at Brandeis, COSI 11A[3] was the most attended course at 257 students, beating out the introduction courses to majors that were previously the most popular (e.g. Biology, Chemistry, Economics).

Figure 1: Enrollment in COSI courses



[4] [5] [6]

COSI 11A is a standard Introduction to Programming type course, with no pre-requisites for joining. It uses Java as a programming language to teach the basic concepts of programming, such as methods, variables, iteration, flow control, arrays (including nested arrays), searching/sorting, the basics of objects, and recursion, as well as I/O through stdin/stdout and files. The lectures themselves consist of the professor, most years Professor Antonella Di Lillo, going through a slideshow about the specific topic, and there would be standard programming assignments that accompany the lectures, reinforcing the topics. The grade consists of these programming assignments, three exams, and a participation grade. The class is extremely popular, and was the second largest course at Brandeis Fall 2016 with 257 students in two sections.

A large number of these students are incoming freshman who are interested in continuing the computer science major. For them, this course provides a very solid backbone to continuing the major. The practice

[3] formally titled COSI 11A - Programming in Java and C, all though C is not covered.

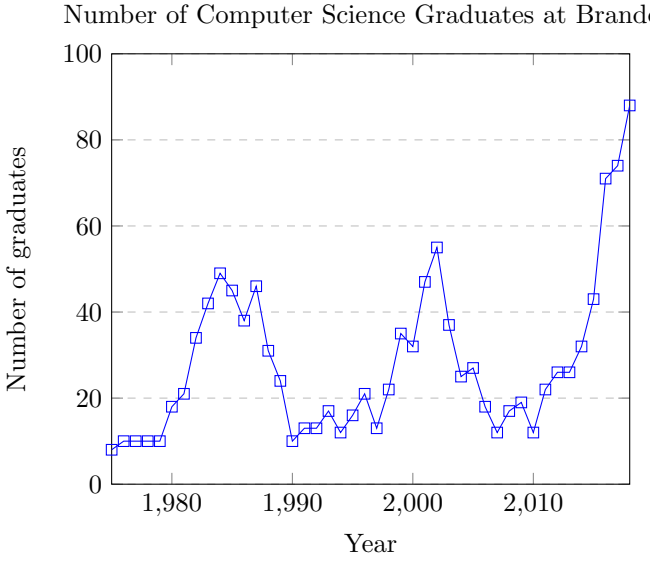
[4] Data from www.schdl.net, which gets the data from the Brandeis registrar. Data only goes as far as Spring 2006, however

[5] One reason for the decline in COSI 11A Enrollment at the end is a stricter cap on the number of people allowed to enroll in the course

[6] Since most of the classes are offered per semester, the year is the year of the fall of that year. Meaning a class that was offered Spring 2010 will be stacked under 2009. One exception to this is COSI 21A, which was offered in the Fall and Spring of the 2016/17 year, hence the jump in enrollment

[7] 2017 and 2018 are projected

Figure 2: Number of Computer Science Graduates at Brandeis[7]



and grading provided in the programming assignment solidifies the tools given by Java, and teaches students how to write good code (with modular methods, good comments), as well as good Java code. This is especially useful in continuing in the Brandeis CS department, where other courses such as Data Structures, Advanced Programming Techniques, Operating Systems, Database Management Systems, and others expect a solid understanding of the Java programming language.[8]

However, there are a large number of students taking COSI 11A who have no intention of continuing in the computer science major. These may be students who have heard of computer science, and want to see what it is before committing. These curious students could just be using COSI 11A to fill their School of Science requirement at Brandeis. [9] There are also many majors that students largely benefit from learning programming, such as Biology and Physics, where labs on campus require students to run and modify scripts to analyze data, as well as output graphs and CSV files for further analysis. These students might take COSI 11A to further their understanding of what is going on when modifying these scripts, as well as getting the tools for extending the scripts to do do updated and related tasks.

It is my belief that although COSI 11A does a fine job of preparing computer science students for the rest of the major, it does a disservice to the latter group of students, whose needs do not match the goals of the course. As it may be the only course they ever take in programming, the course does not introduce enough or contextualize the material enough to make programming useful outside of the course.

1.1.2 Problems with COSI 11A

One of the biggest problems for all students, but especially those who are using COSI 11A as their only programming class, is the choice of programming language to teach it. Take for example the canonical "Hello World" program in Java (Figure 3).

[8]Even courses that don't use Java (such as Software Engineering in Ruby, or Stat NLP in Python) expect you to know Java, and introduce their programming language expecting students know Java

[9]Brandeis Undergraduate are required to "complete one semester course in each of the four schools of the university: creative arts, humanities, science and social science."

Figure 3: Hello World program in Java

```java
// Hello.java
public class Hello{
   public static void main(String[] args){
      System.out.println("Hello World");
    }
}
```

While it may look extremely simple to a seasoned programmer, there is a lot to take in for someone just beginning. In order to not have to completely memorize this structure, the student would have to understand what an object is, what it means to make an object public, what is a method, what is a main method, what is a static method, what are parameters, what "String[] args" means[10], where does it come from, etc... This is not even including the semicolons and brackets, which aren't inherently obvious at first sight, or easy to remember in subsequent programs. Needless to say, this program ends up becoming an incantation for students to get something to appear on their terminal.

Throughout the whole course, students are expected to learn both a cumbersome language, as well as the concepts behind computational thinking. The idea of COSI 11A is supposed to be using Java as a means to learn general programming practices, but many lectures are devoted to the subtle and unintuitive Java syntax, for example the scanner. Another reason Java is not the best tool for teaching programming is the lack of a REPL. In all stages of learning programming, a REPL allows the learner to easily try out a line of code, to see what it does, and interact with it on the way. [11] Unfortunately, Java does not provide a REPL for this type of quick testing, meaning that students have to recreate their file and recompile their code every time they want to see the effects of a small change. The REPL also allows for incrementally coming up with an answer, which is a way that many people like to learn programming. The lack of a REPL discourages learning and discovering things in Java, and makes programming a much more frustrating experience than it should be. There are websites such as `www.repl.it` and `www.pythontutor.com` which try to make a REPL for Java, but it just doesn't work as well as something native.

Figure 4 shows an example of what a student could do in Python to figure out how to write the code to get the middle three elements from a list.

Finally, for the students who are taking COSI 11A as their only programming course, learning about Java and its intricacies is not a good use of their time. Java is much harder to use as a general purpose programming language, being mostly used for Android applications and enterprise software instead. In my opinion, for students wanting to start their own projects, it is much easier to write out your ideas in Python, as well as to find a library that is easy to install and use. Additionally, many labs on campus tend to use languages such as Python or R, rarely languages like Java.

Unrelated to the programming language, COSI 11A is taught and organized with computer science majors in mind. The topics are introduced without much background on how they can be used outside the class, with no context in the overall usage of the subjects taught. Additionally, there is a lot of time spent on the intricacies of these topics and all of the edge cases, which are useful for computer science majors who

[10] Some graduating CS majors still don't know what the String[] args is for, as they've never used it.

[11] Here is an interesting blogpost on that opinion: `http://www.programmersparadox.com/2008/01/22/your-first-programming-language-needs-a-repl/`

Figure 4: Example usage of REPL to find middle three elements of a list in Python

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> x = [1,5,4,7,5,3]
>>> x[len(x)/2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices, not float
>>> len(x)/2
3.0
>>> x[len(x)//2]
7
>>> x[len(x)//2-1]
4
>>> x[len(x)//2-1:len(x)//2+1]
[4, 7]
>>> x[len(x)//2-1:len(x)//2+2]
[4, 7, 5]
```

will need to have a mental model of these things for future computer science courses, these skills are not transferable to people who will not be coding their own data structures.

In addition to helping the students who want to take a course to get a good background with programming, having a course for non-cs majors will also allow the computer science department to have a smaller course for those who are interested in using COSI 11A as a stepping stone to the major, and be able to focus more of their energy on those students, as opposed to trying to make everyone happy.

1.1.3 Background of COSI 2A

In the past, COSI 2A has been a course at Brandeis for non Computer Science majors, titled "Introduction to Computers." It provided a background to what computers were, how they worked, and other general information about topics in Computer Science, generally related to the professor's interests. Many people take the course in order to fulfill their school of science requirement, but it has been criticized for not being substantiative enough.

This semester, Professor Tim Hickey, who was scheduled to teach the COSI 2A course, decided to make the class an introduction to programming in Python for non Computer Science majors, as there was a high demand for a programming class that was not as rigorous as COSI 11A, both from the students, and from the professor of the class who was faced with extremely large classroom sizes.

After finding this out, I proposed to Professor Hickey that he should turn the second half of the course into an introduction to practical programming, since it answers many of the problems stated previously. As someone who has been a teaching assistant for COSI 11A, as well as being very involved in the Computer Science club at Brandeis (BITMAP), this problem was one that I was very interested in solving. I wanted to use the COSI 2A course as a way to see if we can create a class in Python that is suitable for teaching

programming and computer science concepts to non computer science majors, in a way that was both rigorous, and useful for them after finishing the course.

Using Python already solves a lot of the problems created by making an introduction class in Java. Python has an extensive collection of libraries and tools that are easily available for download, and very easy to just install and use. For example, people from science majors can use libraries such as numpy, matplotlib, scipy, and scikitlearn to run experiments, as well as collect and analyze data. They can also use the scientific notebook tool called Jupyter to present the code along with the data created in a cohesive way. Other things that are simple to do in python (just by intsalling a single library) include creating websites, making games, doing sentiment analysis on text/images, analyzing music scores, loading CSVs, and many more possibilities. As of March 23rd, PyPI (the python package repository) lists 101337 different packages available to install, with practically anything you need to do being a simple installation, and reading the documentation to get started. The goal of COSI 2A is to teach students basic python functionality, and then teach them how they can find packages that solve their problems, as well as how to run them.

Another problem it solves, as presented before, is that Python contains a REPL. The tool was discussed previously, but pedagogically, this lets students explore and figure out things for their own, instead of having to create a new file every time they want to try something new.

Finally, Python is a much easier language to get started programming with, as the language was designed with the principle of least surprise. Much of the syntax is designed to be very readable, and intuitive for beginners. This makes it extremely effective as a programming language used for "real life" matters - as less time needs to be spent on the core functionality in order to get more interesting possible projects from students.

1.2 Goals and Research Questions

My research question is as follows:

> Can we make an introduction to programming class in an interdisciplinary setting that is accessible to beginners and non COSI majors, as well as gives students more practical skills, while still learning the fundamentals? Can this be used as a replacement for introduction to Computer Science for all students?

As this is only a senior project, with limited time to complete and analyze it, in addition to not having a very good control for the course, I am instead proposing that the results found from this course serve as a motivation for creating a thorough study on this type of course for non majors, and how they compare to what is currently being offered. In addition, I will propose that students from this course are tracked through Brandeis, especially if they end up doing the computer science major, to see how they fare against students who haven't taken the course.

2 Demographics

2.1 Recruitment

Since Professor Hickey and I were making a new course from what was in the course description and what has been offered previously, we decided on doing an extensive advertisement of the course. This also allowed us to get a much more diverse set of people taking the course, from different academic backgrounds, who

normally wouldn't want to take a course in programming. Since COSI 2A is an experiment in getting non CS majors programming, it was extremely important to make sure that we get these types of students in the class, so we could see the effectiveness of this type of class.

To do the advertisement, I sent a personalized email to several of the departments at Brandeis who I would expect could gain a lot from leaning programming. Figure 5 is an example sent to the Biology listserv.

Figure 5: Invitation to Biology Department

> Hey everyone!
>
> Are you interested in learning how to code, and applying it to your interests/major? Professor Tim Hickey and I are redesigning the COSI 2A course to be an Introduction to Real Life Programming in Python, for non CS Majors.
>
> The first seven weeks or so will be covering the basics of programming, while the rest of the course will be learning about how to apply programming to real world problems (e.g. your research or own personal use).
>
> It should be a fun time, and we are looking for people outside of the computer science major who are interested in what programming is, and how they can use it. We want a diverse class with people of many different interests, so that we can explore a wide range of the interdisciplinary uses of programming (e.g. Biology, Music, Psychology, Social Science, etc), and have a good range of final projects at the end.
>
> It would be great to have Biology students join the class, as there's a lot people can do with biological data once they know how to visualize/manage it in Python. I know a lot of lab jobs also require knowing programming, and it's a much more pleasant experience once you have a good grasp of the fundamentals.
>
> Let me know if you have any questions!
>
> -Arya Boudaie

Similar emails were sent to many other department email lists, club email lists, and even class Facebook groups. This advertising campaign was extremely successful, as by the end of the add period, COSI 2A was one of the largest classes at Brandeis that semester, especially for one that didn't fill any major requirements (See Table 1 and Table 2). Many students were filled in on this after reading the emails as well (see Table 10) After the drop period for COSI 2A, we had students fill out a demographics survey, so we could know where the students were coming from, how they heard of the class, and what their backgrounds were.

2.2 Demographics of Students

One of the first things we noticed in the data for the survey is that several of the students in the course had already taken higher level computer science courses at Brandeis, and were just in this class for an easy A, or a relaxed way to learn Python. Unfortunately, due to a fluke in the registrar's data, the course was also listed as a Computer Science major elective, despite its intent for non CS-majors. Out of the 149 people in the course who took the survey, 34 of the students reported taking courses in the CS Department. As the

Table 1: Most popular courses at Brandeis Spring 2017 (by single section)

Course	Number Enrolled	Required for Major
COSI 2A	158	None
NPSY 11B	143	None
REL 151A	135	None
COSI 12B	108	Computer Science
MATH 37A	106	None
ECON 10A	104	Economics
BIOL 17B	101	None

Table 2: Most popular courses at Brandeis Spring 2017 (adding sections together)

Course	Number Enrolled	Required for Major
BIOL 15B	250	Biology
CHEM 18B	173	Chemistry
COSI 12B	171	Computer Science
ECON 20A	162	Economics
COSI 2A	158	None
CHEM 29B	147	Chemistry

goal of this thesis is to study the effect of the course on people with little to no programming experience, I have discarded those people from the data, leaving 115 people in the study. Note that this does not include people who have taken the Introduction to Computer Science course, but mentioned in the survey that they did not do well in that course, and wanted a second chance with COSI 2A.

Of these 115 students, 63 reported being female, 51 reported being male, and 1 reported being non-binary, making the class 56% non-male. In terms of race, 48 students reported being Asian, 40 reported being White, 16 reported being Black, 10 reported being Middle Eastern, and 1 reported being "other." In addition, 5 reported being of Hispanic or Latino origin (with 9 preferring not to say). Other notable statistics include the large number of students with absolutely no programming background, the large number of first generation college students, and the number of different majors represented in the course.

The survey demographic information is displayed in full in this section's tables.

Table 3: Gender Breakdown

Gender	Number of Students	Percentage of Total	Percent of Total (11A)
Male	51	44.3%	52.9%
Female	63	54.8%	41.1%
Non-Binary	1	0.01%	0.0%

Table 4: Race Breakdown

Race	Number of Students	Percentage of Total
Asian	48	41.7%
White (Non Hispanic)	35	30.4%
White (Hispanic)	5	4.4%
Black	16	13.9%
Middle Eastern	10	8.7%
Other	1	0.01%

Table 5: Math Background of Students

Math Background	Number of Students	Percentage of Total
Less than Precalculus	6	5.2%
Precalculus	19	16.5%
Calculus in HS	35	30.4%
Calculus in University	34	29.6%
Beyond Calculus	21	18.3%

Table 6: Years in School

Class Year	Number of Students	Percentage of Total
Freshman	32	27.8%
Sophomore	26	22.6%
Junior	28	24.3%
Senior	28	24.3%
Graduate Student	1	0.01%

Table 7: Programming Background of Students

Programming Background	Number of Students	Percentage of Total
None	62	53.9%
Very Little	22	19.1%
Coded on my own	8	7.00%
Took a class, not serious	13	11.3%
Took serious class	4	3.5%
Other	5	4.4%

Table 8: First Generation College Students

First Generation College Student	Number of Students (2A)	Percentage of Total (2A)	Percentage of Total (11A)
Yes	26	22.6%	9.8%
No	86	74.8%	86.2%

Table 9: International Students

International Student	Number of Students (2A)	Percentage of Total (2A)	Percentage of Total (11A)
Yes	39	33.9%	25.5%
No	76	66.1%	74.5%

Table 10: Found out about course through

Found Course Through	Number of Students	Percentage of Total
Course Directory	59	51.3%
Another Student/Friend	49	42.6%
Department/Club Email	33	28.7%
Facebook Post	10	8.7%
Professor's Recommendation	6	5.2%

Table 11: Departments Students heard about course from

Dept. Email From	Number of Students
Business	20
Computer Science	15
Biology	7
BITMAP (CS Club)	5
Chemistry	4
Psychology	3
Sociology	2
Economics	2

Figure 6: Majors represented in the course (35 unique)

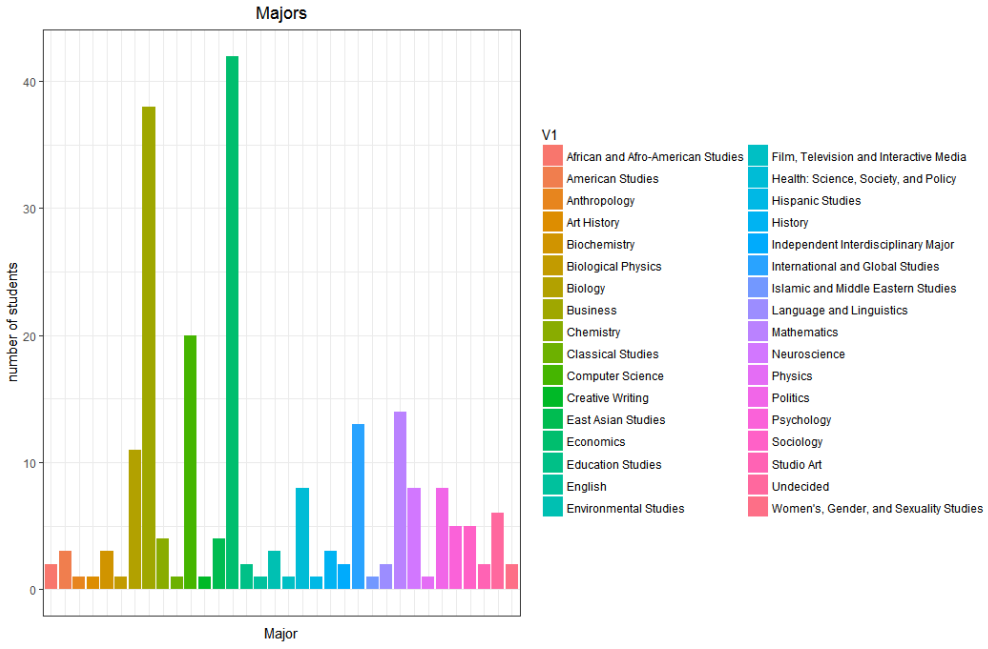


Table 12: Considering CS Major/Minor

Considering CS Major	Number of Students	Percentage
Yes	21	17.1%
Depends on How Class Goes	38	31.0%
No	64	52.0%

3 Overview of Course

The course we designed was split into two units. The first unit was for rapidly learning core python functionality, while the second unit was using that core python functionality with external libraries and APIs to extend the power of Python. While I was not able to change the course title or description due to bureaucratic issues, here is the ideal course description for COSI 2A:

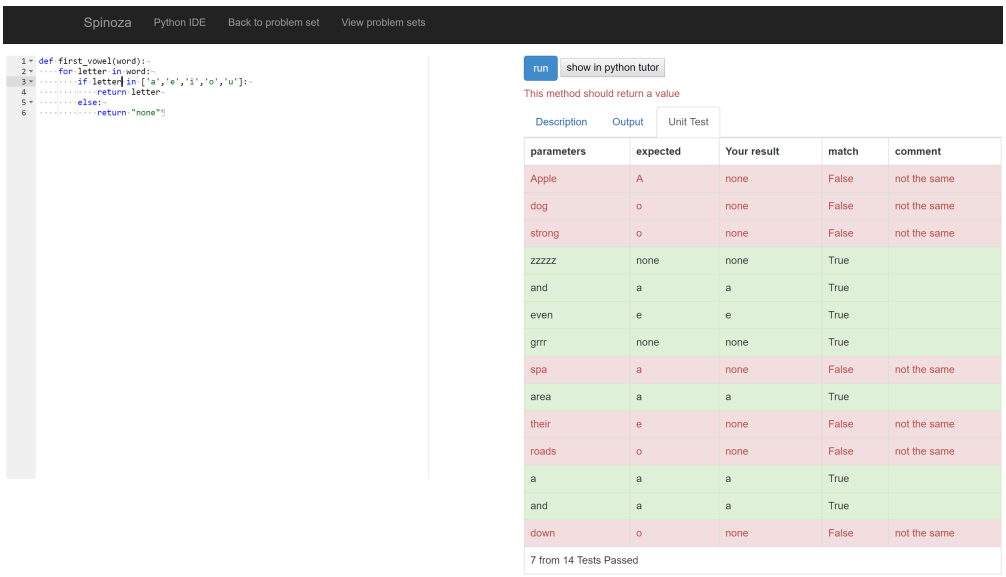
> This newly designed course will cover the basics of programming and the type of thinking behind writing computer programs. This course will use Python, a programming language that is excellent for beginners, but is also robust enough to run companies such as Youtube, Instagram and Pinterest, as well as countless research labs in various disciplines. The course will start off with the basics of programming, and then branch off into the many ways you can use Python, such as: building websites, analyzing sentiment in text, or developing the skills needed to solve your own academic problems. We want students of all majors and disciplines to enroll and bring their own perspectives on the different ways code can be used. This course is a good standalone course, but will also prepare you to take future COSI courses.

3.1 Unit 1 - Rapidly learn core Python

Professor Hickey and I decided that the course would be best designed in two parts. The first part would be an introduction to programming and computational thinking in general, similar to any regular introduction programming class. There were a few differences with how we ran the class though.

For example, one of the main teaching utilities we used was a website called Spinoza, developed by a PhD candidate at Brandeis, Fatima Abu Deeb. This website allows instructors to create practice python problems for students to solve. All they have to do is define the function, and then when they click run, several unit tests will pop up, telling them which tests they got correct and which were incorrect.

Figure 7: Example of a Spinoza problem (return the first vowel), with unit tests done on an almost correct solution



The first thing this allows us to do is create an expectation of having certain python concepts solidified.

The students are allowed to use any internet resources to solve the problems, mimicking what they would be allowed to use in a real life, but with these resources they should know how to solve problems related to the concepts covered in class. Spinoza also allowed us to collect real time statistics on which aspects of Python most students were understanding, and which ones we needed to make more problems for. Besides Spinoza, students were expected to be comfortable editing code in a text editor, and running it on their own command line.

The classes themselves were also taught in a non-traditional way - instead of having the professor giving a standard lecture and taking questions from students, the questions are all asked on an online forum in the course platform TeachBack, for everyone to see and collaborate, so that the professor can present without being interrupted. TeachBack also has a method for the professor to ask questions; either those with a correct/wrong answer, or just a question asking students if they are lost or not, and in that instant be able to tell where the class is at that moment. This, combined with the Spinoza website, allowed for the professor and I to know where the students were and what they understood at any given moment during the lesson. TeachBack is a general software for managing courses and lessons developed by William Tarrimo at Brandeis University for his PhD dissertation, and it aided very well in managing a class that was so large.

There were two quizzes during the course to reinforce the basic python concepts that were taught during the class. The quizzes were in class, but done online on the spinoza website. The students were allowed to use any online resources to help them solve the problem (including their own notes and the notes on the course website), as long as they didn't use any chat programs to ask other people to solve their specific problem. This environment is to make sure that we're testing the students ability to solve problems, not memorize things in Python.

The goal of this part of the course is to give the students a solid understanding of core Python skills, so that when learning about how to write real life code, they have a solid understanding of how to write good python programs.

The course goals for this section are as follows:

- Be excited about learning how to code (and learning more outside this class).
- Be able to solve simple problems with methods, loops, lists, etc. Have a leg up in taking COSI 11A (or just have them prepared for COSI 12B).
- Be comfortable with discovering, learning, and using built in/external libraries, modules, and even other languages on their own PAs
- Have a good mental model of how Python works. Being able to know what the code will return by looking at it.

3.2 Unit 2 - Solving Real World Problems

3.2.1 Goals

The goals for this unit were as follows:

Learning Goals - by the end of the course, all students:

- Should be able to read and understand documentation of internal python libraries.
- Should be able to download and use and use external libraries from pip.

- Should be able to combine knowledge of basic python with libraries to use them well.
- Should be able to use an online API - whether it has a python library or not (using the requests library). How to make a get request, post request, etc...
- Should be able to make basic website in flask, with HTML and python logic. Should have support for database data. Could make own API - get and post requests. [12]
- Should be able to upload code onto Github, make commits, collaborate with others, etc... Should be able to properly document code for portfolio (READAME, blog post, video, etc), should make requirements.txt file. [13]
- Should be able to find resources to solve a problem they are having themselves.

Contrary to other "real-life" programming courses offered, such as programming boot-camps, we decided to strictly only start talking about the real-life aspect of programming until the students had a good understanding of the basics of the Python programming language. This was to mitigate the problem of students just memorizing random syntax that seems arbitrary while using Python libraries, instead of understanding what methods and functions are, and how to use them. The idea with this unit is to get students to extend their understanding of the Python programming language with additional libraries - first teaching them how to use specific libraries, with the goal to eventually have them learn how to use libraries on their own.

3.2.2 Programming Assignment

Instead of having discrete sections of Unit 1 and Unit 2, the professor and I decided to slowly introduce the concept of useful programming to the students, while still discussing core Python fundamentals. The first step was assigning a programming assignment that allowed for the students to be creative in their design of the programs.

Since the students worked on Spinoza to learn core Python skills, writing short functions to solve tasks and pass unit tests, we created this programming assignment to guide the students into writing programs for their own purpose. The assignment was to create four functions, that can solve any problem that the student finds interesting. The only limitation was that the problems they solve couldn't be too simple, and they had to cover loops, if statements, booleans, and calculations. For each function, they also had to include a docstring [14]. with the program's description, as well as several unit tests that could be used to test if the function works.

After coding the function, they also had to make an interactive version of the script that gets arguments from the command line using the input() function, convert the arguments into the appropriate data types, call the function, and then print the return value of the function in a nice way. [15]

The goal of this assignment was to not only allow students to be creative, but also to enforce good coding practices, as well as give students the courage to run programs on the command line, and give them an idea of how to make an interactive program, to be used in a lab or somewhere related. It was also meant to spark creativity for the final project (to be discussed later).

[12] While we got to basic flask knowledge, we didn't get to using a database

[13] Unfortunately, we ran out of time and could not get to Github.

[14] documentation string under the program, which is read when the help() function is called on the function

[15] The interactive part of the function was required to be in a block that begins "if __name__ == '__main__':" - which means "only run this code if it was run directly, not imported." This is a confusing pattern for Python beginners to learn, but is also a common pattern seen in Python code, so we felt it was necessary to require that little step, despite the confusion it caused.

Figure 8: Sample program for PA1

```
def rectangle_area(width, height):
  """
  Takes in the width and height of a rectangle,
  and returns the area.
  Example answers:
    rectangle_area(10,10) = 100
    rectangle_area(5,7) = 35
    rectangle_area(1,2) = 2
  """
  return width*height

if __name__=="__main__":
  print("This program will calculate",
        "the area of a rectangle")

  # Gather the inputs for the functions
  width = int(input("What is the width "))
  height = int(input("What is the height "))

  # Calculate the answer using the function
  answer = rectangle_area(width, height)

  # print answer function gives back
  print("The area of your rectangle is",answer)
```

To mitigate the stress that programming assignments normally give, we offered students feedback, as well as the chance to correct their assignments for 100% of the credit back, to encourage them to not just do something easy for points.

Figure 8 shows an example of the structure of a solution (though the content would be too simple to count for credit).

3.2.3 Jupyter Notebooks

Another thing we did to introduce students to real life programming was start lessons with Jupyter notebooks [16] containing the text of the lesson, as well as sample code to get students started, and with questions to fill out. Jupyter notebooks are a scientific notebook for many programming languages, which allow you to have both rich text and runnable code blocks in a "notebook." These notebooks are also easily saved as HTML pages or PDF files, making them a perfect tool for people to report the results of code they have written, including the code itself.[17]. The ease in terms of presentation and allowing supplied code to be instantly run made using the Jupyter Notebooks a good choice for introducing the class to new libraries and

[16] http://jupyter.org/

[17] Here is an example of a Jupyter Notebook I created to present an implementation of a ride sharing algorithm for a final project for a Game Theory course: `http://www.cs.brandeis.edu/~arya/cost-allocation.html`

packages which may have confusing syntax and ideas to start off with. The Jupyter notebooks are also a good tool to introduce to students, as they could use it to display the results of their code along with the code itself. Besides the final project in the course, it is also a useful tool for outside of class, especially for lab reports (for science students), or general research projects. Designing these notebooks was a significant portion of my work for implementation of the COSI 2A course.

The design of the notebooks was inspired by the POGIL design of lectures. The POGIL [18] design principle for lessons is to create worksheets that inspire students to inquire and discover for themselves how the things they are learning about work, instead of being told and lectured at by the professor [1]. This process worked perfectly for learning about external libraries, as we wanted the eventual goal for students to be able to discover their own built in/external libraries, and figure out how to use them without a pre-built lesson around the library. This style also fit in our lecture, as we already had the idea for a classroom where students would code during the class, instead of just listening to the professor.

The Jupyter notebooks were originally meant for the libraries, but because they were very popular for having notes for Python, we ended up making some for core Python functionality as well, such as list comprehensions, and dictionaries.

The Jupyter notebooks in general were great for this sort of lesson, as they allowed the material to presented at the same time as the actual code, and it allowed for students to work on the problems right there, as well as have written notes of what has been previously covered. The only problem with the implementation with Jupyter notebooks makes it hard to grade, as there was no way to automate checking the notebooks for the number of students we had. The way we got around this was having students defining the functions in Spinoza (with unit tests), or having them paste their code onto TeachBack and testing it there. [19]

3.3 Methods on Objects

After a notebook introducing how to use Jupyter notebooks, our first notebook was one about discovering advanced string methods, which you can view in Appendix A. To summarize/comment on, it starts with the goals of that notebook: what the student should expect to know by the end of the worksheet, and what they should seek help understanding if they don't understand it by the end of the class. It then reviews the basics of strings that the students learned about in previous classes, and that will be relevant in this lesson, so that the students don't get stuck on something they don't remember. The review also contains a review on loops, accumulators, and functions. The function reviewed, "count_As," is a function that takes a string as input, and returns the number of times the letter "a" appears in the string. After the review of the function, the student is asked to modify the function to count how many of any vowel appears in the string, to make sure they are understanding the basics of what has been covered. For testing this function, I also gave the entirety of the text of Romeo and Juliet as a string, which served both as an interesting data set, and a good way to ask a question on the TeachBack forum and see how many people got the right number of vowels. It also served as an introduction to opening and reading files, something that we would introduce more formally later.

After this, the purpose of the lesson, which is to discover built in methods, was introduced. We introduced the "count" method on a string (which was the function that was written above in whole), and then asked

[18] Acronym for Process Oriented Guided Inquiry Learning, learn more at https://pogil.org/about

[19] You can view some of the Jupyter notebooks in the Appendix, and they are also available for download at `https://github.com/misingnoglic/2a_tutorials`

students to submit on TeachBack what they think it does. This was meant to demonstrate one way to try to figure out what a method does - by guessing based on the name, and what it outputs. The first one we chose, count, was fairly straightforward, and a good option for a first function. We then asked them to rewrite the "count_As" from the top, so to demonstrate they know how to use the .count() method on strings, and so they can appreciate the usefulness of the built in methods.

After this, we introduced another way to find out about built in methods - guessing what a method is called based on what it does. We introduced the s.lower() method, and asked students to guess what other methods there would be, and post on TeachBack. I chose the lower() method, as it has a very obvious connection with the upper() method, as well as other methods on the string related to case. This wasn't an extremely important skill to cover, but definitely something worth mentioning.

Once the POGIL experimentation was done, we introduced how to find the methods on an object such as a string (using the dir() method), and then how to find out what each method does (using the help() function on the method). After this explanation, we tested their knowledge on how to use these function, by asking them to upload to TeachBack what is the functionality of several of these methods.

3.3.1 Text and CSV files

The next "real" topic I wanted to cover in the course was reading and writing from files. This seemed important to cover as most Computer Science courses cover it, and it is a good way to create code that is useful for other people. In addition, as many tasks involving analyzing, reading, and compiling data involve CSV files, we decided to make the focus of teaching about files revolving around CSV files, and the CSV library in Python that makes it easy to read and write CSV files.

The first thing we did was create a Jupyter notebook about reading and writing files (B). This notebook starts by going through the basic syntax of reading files, going through each step of turning a file into a string.

After going through the basics, we introduce a text file containing the entire contents of Romeo and Juliet, and explain how the basic tools for reading and iterating over files could be used to read a large text file such as the entire contents of Romeo and Juliet. We also discuss writing to files.

In the next Jupyter notebook, we explain how to use the CSV library to go through a CSV file in Python, reading and analyzing the data (**??**). CSV files could be traversed like a normal file (splitting on the commas), but the CSV library makes it a lot more easy and manageable, and a very effective tool for bringing out of the course. The notebook starts with talking about what CSV files are, and what they're used for. It then talks about how you could parse a CSV file if there was no CSV library. After this, I introduced the syntax for the CSV library, reading files using both the reader (returning a list of lists) and the DictReader (returning a list of dictionaries with the headers being the keys).

After explaining how to read the CSV files, the notebook works through some examples of using CSV files with real life data, such as a CSV file with sales data from Sacramento homes. The notebook explains how to parse this data, as well as get answers to real questions from it, such as the average price of house sold, the most expensive house, the most expensive zip code, and the house that's the most expensive per square foot (all done by writing a for loop and keeping an accumulator or variable to keep track).

The goal of this section was to hopefully give an idea on how coding skills could be used on real life data, to either generate it, read it, or analyze it.

Figure 9: Hello World program in Flask, as well as another page that takes an argument

```
# hello.py
from flask import Flask

# Creates the app
app = Flask(__name__)

@app.route('/') # Routes this function to the homepage
def hello_world():
    return 'Hello World!'

# Takes <name> as an argument in the URL
@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello '+ name + '!'

# run the app
if __name__ == '__main__':
    app.run()
```

3.3.2 Flask

The next thing we wanted to cover in the course was how to make websites - something that was very tangible and real, as opposed to command line applications. Since most programming courses focus on just making command line applications, most students out of these courses have no idea where to go from there to take their skills and make programs that people can actually use. Python has a very popular micro web framework called Flask [20], which makes it very easy to make simple websites with very little code or boilerplate, making it a logical choice to introduce to the students as a tool that can be used to bring an intuitive interface into their programs.

To introduce this section, I created a slideshow [21] to explain how each line of the Hello World program worked (Figure 9), as well as how to run it. After, I explained how Flask works, and how it turns a URL into a generated webpage.

Once I explained the basics, I demonstrated how a program could get input from the URL itself, so that web pages could be dynamically generated. An example is in Figure 9, where the second function takes ¡name¿ as an argument in the URL, and then returns a webpage with that name. This was presented in a way to demonstrate how one could turn one of their assignments from PA1 into an interactive website.

The assignment for the class before this one was to read on basic HTML, so after showing how to get arguments from the URL, I demonstrated how one could use HTML to generate a nice looking website, rendered on the fly. This is done by generating an HTML template, with extra code in it to take information from variables, or to loop through passed in lists. The example from the class was to make a webpage that displayed a list of factors.

[20] Documentation found at `http://flask.pocoo.org/`

[21] Link: `https://docs.google.com/presentation/d/1DC0_1RIvmLOvwqQLsmDALre1BHgkSAkkpe9V2tBbdwc`

Figure 10: Function to render factors HTML

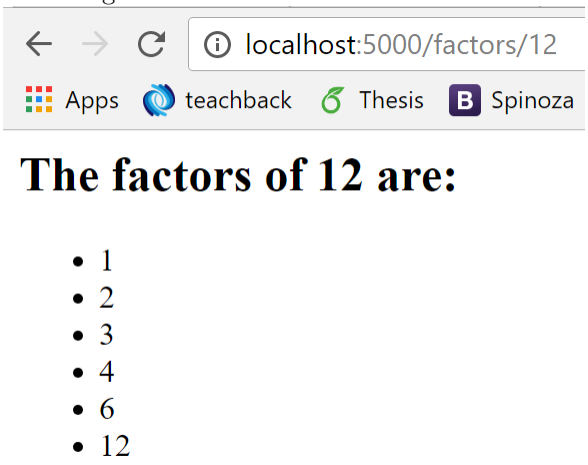
```python
# hello.py
def factors(n):
    """
    Program that takes a number (n), and returns
    the list of its factors (numbers that divide into n)
    """
   answer = [ ]
   for i in range(1,n+1):
      # if i is divisible into n, add it to the list
      if n%i == 0: answer.append(i)
   return answer

@app.route('/factors/<n>')
def factors_display(n):
   factors_list = factors(int(n)) # Convert it from string (URL) to int
   return render_template(
      "factors.html",
      number=n, # passes the value 'n' as the variable "number" in the template
      factors=factors_list # passes value of "factors_list" as variable "factors" in template
   )
```

Figure 11: HTML for Factors Page

```html
<html>
   <title>Factors of {{ number }}</title>
   <body>
      <h2>The factors of {{ number }} are:</h2>
      <ul>
         {% for item in factors %}
            <li>{{ item }}</li>
         {% endfor %}
      </ul>
   </body>
</html>
```

Figure 12: Rendered Factors Website



After this basic introduction to Flask, the hope was that anyone who was interested could look up more on how to use Flask to create the kind of website they were looking for, using the slideshow as a stepping stone to getting there. It was also our hope that students would use Flask for their final projects, to have a more interesting interface than a command line application.

3.3.3 APIs

The last Jupyter notebook I created was one about accessing APIs available online, to be able to send data to one and get meaningful results back. One reason we wanted to introduce APIs to the COSI 2A students was that the services offered by APIs allow code to do very interesting things, just by accessing the API's endpoints and receiving the data back. For example, APIs can let Python programs check the weather, send emails and texts, do sentiment analysis on text, and even get information on Pokemon. Additionally, a lot of real life coding involves accessing API endpoints, sending data to APIs, and working with the data received from the API. These APIs would allow students to extend their Python programming skills by leveraging the abilities of the API, and the power they get from the APIs is definitely one of the more interesting things about real life programming.

For the notebook (D), we decided to use the Indico API [22], which is an API from the machine learning startup Indico. The API is mostly used for sentiment analysis on text, though it can also do things like political analysis, or seeing what objects are in an image. This API was chosen as it has a relatively easy installation process, and very interesting data that can come out of it. For example, with just three lines of code, the program is able to send it a piece of text, and get back a number that describes how positive or negative that text is.

Besides the easy starter code and interesting practical applications, Indico also has extremely good documentation, with easily explainable method calls, as well as what the expected inputs and outputs are. The documentation comes with pre-generated code, depending on your API key, which makes it easy to just copy their code, and modify it match your specific use case. Additionally, the free tier of their platform allows for thousands of queries a month, which is plenty for a student who is just trying out the API to see what it is like.

[22] More information at `www.indico.io`

Figure 13: Sample Indico library code. Sentiment is 0-1, where 1 is positive, and 0 is negative.

```
# indico_test.py
import indicoio
indicoio.config.api_key = "your_key" # replace with your key from indico.io
result = indicoio.sentiment_hq("Today was a very good day!")
print(result) # 0.9157847166
result = indicoio.sentiment_hq("Our presentation was a disaster")
print(result) # 0.0328917056
```

The notebook starts off by explaining what an API is, and what they are used for. After this, it introduces the Indico API, how to install it, and how to set it up with your secret key. After this, it discusses basic commands to get sentiment data from the API (Figure 13). The notebook then discusses another functionality of the indico API, one which takes a large block of text, and returns a dictionary of keywords from the block of text, as well as a percentage of how likely they are to be important to the text. The "political sentiment" API is also discussed, one which takes a block of text, and returns the possible political alignments of the writer of the text. The students are then led to go to the documentation of the API [23], to try and find out what else they can do with the text.

After this, we hoped that students with ambitious ideas for a final project would be able to find an API that matches their goals, and be able to read the documentation well enough to understand how to run it.

3.3.4 Final Project

Throughout the course, it was suggested that the course would culminate in a final project, in which students were expected to create something unique and original using what they have learned in class, hopefully leveraging external libraries and APIs. After the first programming assignment, students were asked to create teams of 3-4 students to brainstorm an idea for the final project. This email was sent out to motivate the idea of the project to the students.

> Hi everyone,
>
> With your first PA, you got a little taste of what it was like to make your own Python projects.
>
> For your final project, you will be working in a group of 3 students to make your own large project, that does something interesting computationally, as well as something useful. You should be solving a problem that you're actually interested in, and that you would like to have a program to solve it for you.
>
> Of course, we haven't talked much about creating programs for use in real-life, but you will get a better idea of what you can accomplish as the class goes on. In particular, we will be talking about:
>
> - How to get data from online repositories, to use in your programs.
> - How to turn simple Python scripts into websites.
> - How to read and write from CSVs

[23] https://indico.io/docs

For Monday, you should form a team with people who have similar interests to you, and try to think of a general idea of the problem you all want to solve. It doesn't have to be specific yet, but you should know what the general area you're going to tackle is. If you need help, you can always email the TAs who have experience in what you're trying to do, and ask them for advice. You can also ask on Piazza, and I'm sure someone will come up with a good area for you to think about.

One good way to make a very successful project is to use your experience outside of computer science: for example, if you are a biology major, you can leverage your biology and programming knowledge to make something extremely interesting. If you are interested in graphics, one thing you can do is visualize data.

To get ideas, you can always Google for things like "python library ¡subject¿"

For example, python library music gets me a lot of interesting results.

You can also search this website: https://www.programmableweb.com/

Try to work in groups with similar programming experience. The grading is going to be largely based on how much work you put into the project, so if you join a group where people are already used to programming large projects, it will be hard to contribute something meaningful. By picking a good group with similar interests to your own, you can ensure that you'll have a fun and productive time working on your project, as opposed to something you're miserable working on.

To look for teammates, we recommend you use Piazza, and click on the "Search for Teammates" button. With this, you can post a little about yourself, and look for others to work with. If you're two people looking for a third person, you can also post about that. We suggest you put information such as your major, what kinds of ideas you have/what kind of problems you want to solve, and what programming experience you've had previously.

Let me know if you have any questions, but otherwise, I hope you have a good weekend!

-Arya

The purpose of this email was to motivate the idea of a final project, and to give the students the background necessary to come up with an idea, since at this point APIs and making things for the real world had not been discussed. At the same time though, we wanted students to get creative with their ideas, since we wanted them to create a minimally viable product (MVP) version of their idea, to be able to demonstrate their coding abilities.

After students had come up with teams, and brainstormed ideas which they started code on, we asked them to submit the following.

You are to upload a description of a Minimal Viable Product (i.e. first initial version) of your Final Project. Your description should include:

- How users will run your project (i.e. as a flask website, or as a Jupyter notebook, or using the terminal)
- Exactly what users will be able to do and how your app will respond
- How you will divide the work up among your team members (which could include all working together)

As the whole purpose for the project was to be largely in the control of the students, our rubric for the final project was fairly relaxed. All it required was that students use the tools that they learned in class, such as functions, lists, dictionaries, calculations, conditionals, loops, and prompting the user for input. Besides this, it was up to the students to come up with final project ideas on their own, with the help of their peers and TAs.

During the class, several ideas for final projects were demonstrated with all the code available, such as a crazy 8s card game, a fake twitter using a Jupyter notebook, and an Eliza style chat program.

3.4 Final Goal

From the beginning of this experiment, we decided that this project will be considered a success if:

- A large portion of the students make useful/interesting/meaningful projects.
- Students feel as confident in their abilities after this class as they do in COSI 11A (based on the survey I sent out).
- Students have confidence in their general coding ability.
- Students have confidence in their ability to use programming outside of class.
- Students who take this class end up doing extremely well in COSI 11A (if they choose to take it).
- Students who do well in COSI 2A who learn java on their own time should be able to pass out of COSI 11A, as well as do well in COSI 21A or COSI 12B (courses that require 11A)

We will discuss the assessment of these goals in the Results section, as well as the future steps section.

4 Results

After the course, we gave an anonymous survey to all of the students to complete, to see what they thought of the course, filtering out those who have taken this course despite already being Computer Science majors and taken many courses in Computer Science. Summaries of the answers to their questions will be displayed in tables and charts below. Some questions are compared to questions asked in COSI 11A - these answers were received from a similar survey sent out at the end of the COSI 11A course.

Table 13: Yes/No Questions about effect of course

Question	Yes	Maybe	No
Would recommend course to friend	73.0%	19.8%	7.1%
Wish to continue programming	70.1%	18.7%	11.2%
Have clear understanding of how to continue in CS	68.7%	18.7%	12.7%
Wish to take classes in CS department	48.5%	21.6%	29.9%
Wish to pick up CS major or minor	35.8%	17.2%	47.0%
Used external libraries	21.6%	0.0%	78.4%
Used API for project	14.9%	0.0%	85.1%

Table 14: Difficulty of Course

Question (Ranked 1-5)	Average (2A)	Average (11A)
Programming Ability Growth	3.9	N/A
Problem Solving Skills Growth	3.8	N/A
Pace of class (1: Too slow, 5: Too fast)	3.8	3.2
Compared to learning a foreign language	3.2	3.3

Table 15: Ranking 1-5 the effect of the course and materials used

Question (rated 1-5)	Average
Usefulness of Spinoza Problems (HW)	4.6
Usefulness of Spinoza Problems (Class)	4.5
Usefulness of first PA	4.3
Usefulness of Terminal Programming	4.3
Usefulness of Quizzes	4.3
Quality of Course	4.0
Usefulness of final project	4.0
Usefulness of Programming	3.8
Usefulness of Jupyter Notebook	3.8
Enjoyment of learning how to program	3.0

Table 16: Students Ranking Skills 1-5

Skill (Ranked 1-5)	Average
Defining/Using functions	4.5
Doing Calculations	4.4
Making interactive programs	4.2
Creating/Looping over lists	4.2
Searching Google for help	4.1
Creating/Using Dictionaries	4.0
Creating Jupyter Notebooks	3.8
Importing/Using libraries	3.6
Reading Documentation	3.5
Reading/Writing CSV files	3.4
Downloading from pip	3.4
Finding useful libraries	3.2
Creating Flask Websites	2.9

Table 17: Students Ranking the Course Goals 1-5

Course Goal (1-5)	Average
Able to solve simple problems with functions/loops/etc	4.5
Understand Joys/Challenges of Programming	4.3
Know which problems can be solved using Python	4.2
Good idea of what someone can use Python to do	4.0
Excited about learning CS	3.9
Clear Understanding of Python	3.8
Able to solve own problems using Python	3.8
Ability to discover/learn about libraries and use them in real life	3.7
Can succeed in future CS Classes	3.7
Comfortable discovering built in/external libraries	3.7
Able to combine knowledge of Python with libraries	3.5
Have plan for continuing study of CS	3.4
Begin E-Portfolio of Work	3.3

Table 18: Comparing Course to COSI 11A, sorted by Δ

Question (Ranked 1-5)	Average (COSI 2A)	Average (COSI 11A)	Δ
Assignments allowed me to be creative	4.1	3.6	0.5
Good idea of how programming is used in real world	4.1	3.7	0.4
Assignments were fun to do	4.0	3.6	0.4
[Python/Java] made it easy to turn ideas into code	3.9	3.7	0.2
Comfortable learning new language on own	3.8	3.7	0.1
Made me comfortable solving real world problems	3.7	3.6	0.1
Clarified understanding of programming	4.3	4.4	-0.1
Clarified skills behind programming	4.2	4.3	-0.1
Made me comfortable solving small coding challenges	4.2	4.3	-0.1
Made me excited about Programming/CS	4.0	4.1	-0.1
Use programming outside of class	3.9	4.0	-0.1
Accessible to beginners in CS	3.9	4.0	-0.1
Comfortable finding/using [Python/Java] libraries	3.5	3.6	-0.1
Assignments allowed me to demonstrate ability	4.1	4.3	-0.2
Course made me more interested in CS	4.0	4.2	-0.2
Gave skills to teach basic programming	3.7	3.9	-0.2
Comfortable writing code at a job	3.4	3.6	-0.2
Assignments solidified lecture material	4.1	4.4	-0.3
Made me feel prepared for future CS courses	3.7	4.0	-0.3
Good mental model of [Python/Java]	3.9	4.3	-0.4
Encouraged to continue in CS	3.5	4.1	-0.6
Wrote something for use outside of class	23.8%	25.5%	-1.7%

4.1 Successes

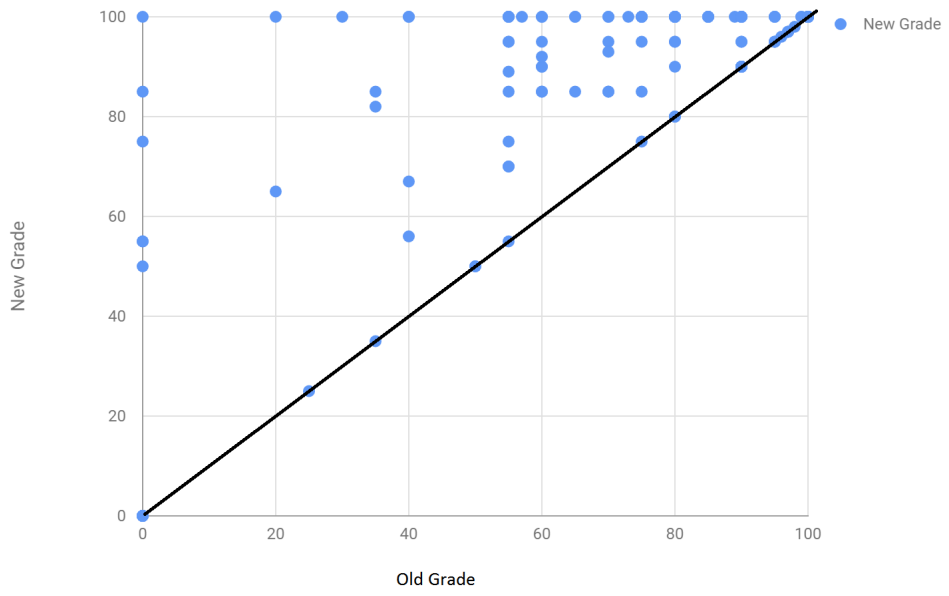
One of the first successes to note is the general diversity created by the advertisement of the course (Figure 6). In terms of majors, we got an extremely diverse group of majors joining the course, which is always good for getting diversity into Computer Science. One of the problems with the CS industry is that the type of people that end up going into the industry tend to have homogeneous ideas and interests, and encouraging people from different majors to learn about programming is one way to overcome that problem. Out of 43 majors offered at Brandeis, 35 of them were represented in COSI 2A. Additionally, in terms of gender, COSI 2A surprisingly has more women than men, which is very rare for a Computer Science course. The Computer Science industry has a big problem in terms of gender disparity, and my hypothesis is that reaching out to groups that don't normally take computer science courses also helps in making the course more even in terms of the gender breakdown. The other demographics of note include the number of first generation college students (over double the number from 11A), and the number of Black/Hispanic students who are generally underrepresented in tech.

In addition, the class got many people interested in continuing their Computer Science education. At the beginning, only 17.1% of students were considering a CS major/minor, with 31% of students saying they would consider it depending on how the class went. Considering the course was meant for people not intending to be computer science majors, and how about 1/4th of the class was graduating seniors, this statistic made sense. However, by the end, 70% of students said they wanted to continue learning programming, 48.5% said they wanted to continue taking CS courses, and 35.8% said they wanted to try to do a CS major or minor (See Table 13. This figure is especially impressive given that almost a quarter of the students are graduating seniors, and would have no opportunities to take future computer science courses. We will have to track the students to see if they follow up on it next year, but it would be very interesting to see how these students do. Finally, the table shows 73% of students would recommend the course to their friend, with only 7% saying they actively wouldn't recommend the course, meaning a significant majority of students were of the impression that the course was high quality enough to continue offering, and that they would want their peers to take.

In terms of questions asked about the quality of the course, most questions were rated with overall positivity. In Table 15, most of the items were ranked with general positivity, with a surprising amount on the Spinoza questions, homework, and quizzes. Besides "Enjoyment of learning how to program" which was split evenly, every other question was generally positive. In addition, in ranking their own confidence in certain skills (Table 16), students seemed very confident in core skills, and semi-confident with the stuff outside of core Python (e.g. reading documentation, reading/writing CSV files). Besides creating Flask websites, everything was ranked above average. When we asked students to rank the course goals (Table 17), they were all ranked positively, meaning that what we originally set out to do was seen as successful by the students in the course.

In Table 18, I compared questions about skills gained in the course to COSI 11A (the regular Intro to programming course), to see how our class compares to that one, in terms of student satisfaction. Most questions were at most only 0.2 points higher for COSI 11A then they were for 2A, meaning that students more or less felt as satisfied with the programming skills gained from COSI 11A (which had very good scores from students that semester). This also makes sense, as the only students who took the 11A survey are the ones who kept up with the course's programming assignments/midterms, while COSI 2A did not have that many dropped students. In addition, students in COSI 2A ranked two of the questions more applicable to the course higher thank the COSI 11A students did (in particular, assignments allowing them to be creative,

Figure 14: Graph of student progress on PA1 (Old v New)



fun to do, and good idea of how programming is used in the real world).

Not related to the survey, the assignments in the course were also very successful. For the first programming assignment, the average on the first submission was a 74.1%. After getting feedback from TAs and having time to resubmit, the average went up to an 88.7%, meaning that many students learned from the comments how to make a well formed Python function and interactive program - something they hopefully take from the class. If it weren't for this generous resubmission policy, I wouldn't think that so many people would take the time to go back to understand (see Figure 14). The black line are the students who stayed at the same grade, but it's very evident that many students got much higher grades than before the submission. The assignments submitted were also very interesting, with programs about statistics, politics, mathematics, biology, games, and more fun topics. I would have wished that more students did these types of programs instead of the bare minimum, but they were still fun and interesting to look at.

We have not been able to fully analyze the final project submissions (due to the nature of this senior thesis), which should be something looked at for further study of this course.

4.2 Things to improve for future implementations

4.2.1 Mentorship

One of the problems we tried to tackle, but failed at was creating a mentorship program for students who were struggling in the course, and needed some extra help. We already offered substantial office hours throughout the week (see Figure 15), with extra thought put into making sure the TAs were balanced in gender, race, and background, there were still many students falling behind in the class who did not go to the office hours. As much as I tried sending emails to students encouraging them to go if they felt lost during a lecture (since

Figure 15: Calender of weekly office hours

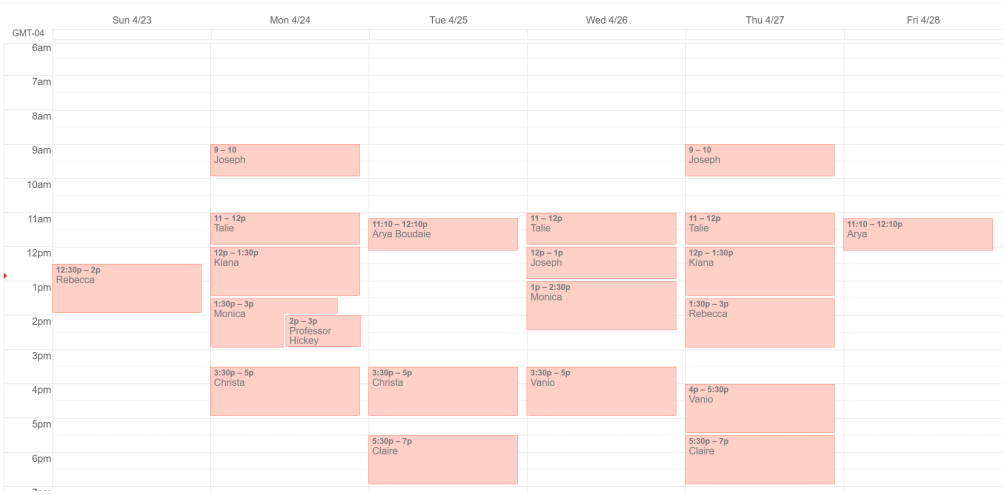
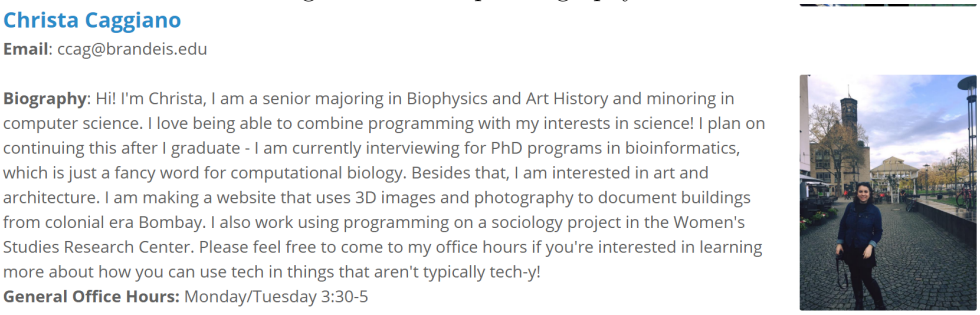


Figure 16: Example Biography of TA



the nature of computer science courses is such that that getting lost in the beginning is very disastrous). We also tried making the idea of going to Office Hours by creating a biography for all the TAs, but most students didn't seem to react to that (See Figure 16 for an example of a biography, meant to inspire students to take programming outside of its normal context). [24] We were also hoping that students would come into the office hours just out of curiosity of things learned in class, but it seemed to mostly just be for getting problems on Spinoza correctly, and for help on the programming assignments.

We also had an online Piazza forum where people could ask questions to the TAs, and get an answer for anything they were confused for during class. This would also have the added benefit of having the answer available to all students, so that they can all get clarified. However, most students did not use the forum for this purpose, instead using it mostly to ask about grades and submission deadlines.

We also tried making a mentorship model where students could sign up to be in a small group (1-5 students) with a teaching assistant for the course, so that they can get one on one attention for the course, making sure they don't fall behind. While its intention was to give struggling students one on one attention, what actually ended up happening was the students who joined the mentorship groups were actually the ones excelling in the course, and who just wanted extra reinforcement of the material they already knew. This was a problem, as TAs who should have time available for struggling students were instead reserved by students already doing well. After a week or so, this program was stopped, and regular office hours resumed.

[24] All the biographies can be found here: `https://sites.google.com/a/brandeis.edu/cs2a-spr17/staff`

By the end of the course, the Professor Hickey also created a morning review session to go over basic Python functionality with students, but the attendance was extremely low given the class size, and the students we wanted to attend did not show up. This program ended up canceled as well.

As of now, we have not found a good way to create a successful mentorship model for students struggling in the course, and further experimentation will need to be done. It is good, however, to look at this course as a model for what has been tried, and how it could be improved.

4.2.2 Atmosphere of Course

One of the main problems through the course, as discussed before, were the group of students who were taking the course for an easy A or major major requirement, and who had already had many computer science courses before. Many of these students cited wanting to learn python as the reason for taking the course, but their placement in the class made the atmosphere of the class much different than if it was a class of just beginners. With these students in the class, the beginners felt more anxious about not getting the material right away, and it made the course a lot more stressful than it should have been. In future implementations of the course, these students should be actively stopped from taking the course, so that the atmosphere could be more beginner friendly.

4.2.3 Size of Course

In the beginning, Professor Hickey and I were extremely satisfied with the number of students enrolled in the class, as it showed a very large interest in the subject material, and would allow us to have a very diverse class. However, the size grew to be very unmanageable, and it became very hard to keep track of students. In a smaller course, it would be easier to get a personal connection with all the students, catch mistakes in their programming style, and guide their final projects. With over 150 students, this became extremely difficult, as the sheer number of students was too much to keep track of. The process of grading one programming assignment was an entire process, where each TA was assigned several students that they had to grade, with not much time for individual attention to be given. For the final project, it would be nice to be able to talk to students and get them to come up with an idea with a TA/Professor who has experience with the pitfalls in programming, but this was not possible, until they had already come to office hours and the damage had been done.

One way to deal with this is to just put a cap on the course, saying that only a certain number of students can take the course. This is the approach taken by the Software Engineering course at Brandeis University, and it allows the professor (Pito Salas) to have a close connection with the students involved. Another solution is we could also have students apply to the course, to make sure that they are dedicated and willing to put the time into the course that is required, and have the creative goals required to take programming and apply it to their real lives. This type of model is used by the directed writing courses at Brandeis, and they have very small and dedicated class sizes as a result. This would also be a good way to ensure a diverse class (as we could use diversity in the application), and that students would have a successful semester in the course. However, this approach might exclude certain people from applying, which would be against part of what the goal of the course is.

One last solution would be to keep the course the same size, and mandate that students go to recitation groups every week for additional lecture, but this would require a very large staff trained in lecturing, which would be difficult, as TAs that would be good for this are also needed in the higher level CS courses at Brandeis. This is the model that is taken by courses such as BIOL 14A (Genetics and Genomics), and it has

worked for them in the past. We could even have students select the TA that they want to be with, based on the background of the TA, and that TA could focus their recitations on the needs of the students. For example, a TA who is a double major in Biology and Computer Science could teach the recitation for people who are interested in Bioinformatics, and the practice problems could be focused like that. This is kind of what we were going for in the mentorship model, but being required might be the extra step to making it work.

4.2.4 Rigor and Pace of Course

Another problem with the course is that the expectations were not set at the beginning. This was in part due to this being the first semester the course was taught, and trying to find a way to manage such a large class. However, in terms of class participation, homework, readings, and office hours, we were not very clear from the beginning, and this set the tone for the class for the rest of the semester. In future implementations, we should be very clear in what is expected from the students for work, so that they can keep that in mind for the whole semester, instead of having the rules change on them all year.

In addition, because we were overwhelmed by the management of the large class, we were not able to make the course as rigorous as we had hoped, in terms of assignments and quizzes. This is partly because during the semester, we realized that many students were behind, and much extra time needed to be spent on the fundamentals, but this seemed to always be true, no matter how many times we went over basics of the language, just because in a course of that size, someone is bound to not be paying attention, or not be explained to in the correct way. In a smaller class, this would not be an issue, and we could move on to more rigorous assignments. Additionally, the size of the class made it very difficult to test for any skills that were not easily testable in Spinoza, namely using APIs and other libraries. In the beginning, we wanted to have several types of programming assignments, including a project for a Flask website, for interpreting a CSV file, and other long form but low stake programming assignments. However, we were only able to have the first PA, as well as the final project, because of the lack of time to go over everything in depth. In addition, we weren't able to cover many of the topics we wanted to, such as Github[25], or have a showcase for the final projects. If we can get the mentorship model right, or the course size lower, this could definitely be something to strive for. Additionally, confidence in making flask websites was remarkably low on the survey (See Table 16, so future implementations should focus more on how to make a flask website, maybe even with a mini project to solidify the ideas of routes and templates. We definitely want to still keep the course friendly and accessible, and not have the atmosphere of a course like Harvard's CS50 (see section 5), but the course would definitely benefit from being less laid back, and more rigorous in terms of assignments and expectations from students.

One way we could cover more material would be to include a mandatory recitation every week for review, just so that the course time could be spent on learning the material. Another way to create more time is to make the course 4 days a week, like a language course at Brandeis, to get an extra day a week for review or alternate instruction. It could also be argued that as programming is a type of new language, it needs the same type of immersion that is created in a foreign language course. This would be a good way to be able to fit in all of the sections we wanted to cover, and still make sure students are getting the most of the material. Finally, we could have a required lab meeting every week, like the software engineering course at Brandeis, to cover more material, have reviews of previous classes, or just give students time to work on programming skills.

[25] Website for managing code with Git, a version control software

4.2.5 Reliance on Spinoza

Students on the survey were very receptive to using the Spinoza program during class to learn about programming. However, this lead to a few problems. One of the problems was evident when one class, we took away the unit tests that Spinoza provides, saying that we wanted students to be able to test the functions themselves, by calling the functions on different inputs, and verifying that the answer is correct. This immediately caused backlash in the course, with many people in the in class forum actually demanding that the unit tests be turned back on. Even with forum posts and in class lessons explaining how the students can come up with their own unit tests, they were still not able to function in Spinoza without the predefined unit tests. In the context of real world programming, it's obviously not good to have people rely on the reassurance of unit tests to write code, as when someone writes code themselves, they need to be self sufficient in creating their own tests, to convince themselves that the code works.

Another problem, as evidenced by the end survey sent out at the end of the course, is that students liked Spinoza, but didn't like the other ways of programming, since they were too open ended. This is a problem, as the students were seeing the Spinoza practice problems as a means to an end, as opposed to just a platform for practicing the basics to later be used in other projects and assignments in class. In particular, a lot of students did not see the point of Jupyter notebooks, and wanted to just do Spinoza instead. A majority of students coming to office hours just wanted help on the Spinoza problems, and it was somewhat clear that they were not getting better at them after getting help from the TAs, so they only came to the office hours to get the Spinoza problems right. A lot of students also thought that functions in Spinoza were completely different from regular Python functions. In the future, care should be taken to contextualize the role of Spinoza - what functions are in Python, and then what the purpose of Spinoza is in their education, as opposed to just letting them infer that Spinoza is the most important part of the course. Again, in a smaller course, this would be easier to do by assigning lots of different assignments besides Spinoza (e.g. submitting the functions as a .py file). Another idea might be to hold off on introducing spinoza, until after students know about creating programs on their own laptops, and testing functions themselves, possibly with the built in unit test library included in Python.

4.2.6 Final Project, Groups, and POGIL

While students were generally happy with the final project, it would have been nice to give the idea more care than it was given. As is the issue with a lot of the class, the size of the class made it very hard to discuss with the students what is an appropriate final project, while still allowing them to be creative. In general, not as many groups as I had hoped decided to use their knowledge from their own majors to create a truly interesting final project. It would have been good to be able to talk to students about their ideas beforehand, and steer them towards projects that are more interesting in the non technical sense. This is also apparent by the percent of students who used APIs and external libraries - despite that being a major part of the second part of the course. In a future implementation, it would be good to help steer groups into making types of projects that will be useful outside of the scope of the course.

For facilitating the group project, there are also a few things that we neglected to do from the beginning. First, we ended up not having enough time to explain how to use Github to students. We didn't want to rush the explanation, as a rushed Github explanation might have made things more confusing for the students than not explaining it at all. However, this meant that students were very unprepared for working in groups, especially over the break, since they were forced to use methods like emailing each other code or putting it

on Dropbox to share code. This led to a lot of problems with some team members having different versions of code, which would mostly be solved by just using Github, which is a good tool to learn in general. In addition, students were just generally unprepared on how to do this kind of team project in code. We should have given them tips on how to work effectively in teams, and maybe even had a sample group project they could have tried working on. Finally, even though we had introduced Jupyter notebooks, we didn't fully go over how to create them until much later in the semester, making them not as intuitive to use for a final project as we had hoped. A future implementation of this course should have a programming assignment in the form of a Jupyter notebook (probably the CSV analyzing one), so that every student understands how to make a Jupyter notebook, and can make one for their final project, and when the course is over.

In addition, it would be useful to have students come up with their groups of three much earlier into the semester. All though it's hard to motivate a final project of this nature before students learn about programming, it would be good to have the group in mind in the beginning, so they can have a lot of time to discuss what they want to make, and how they can improve it, and have the course goal be to complete the project. It will also be good for students to have small groups to work with, as they can help each other understand the material, and go to office hours together.

The groups will also help with facilitating the POGIL type of course we wanted to make. As POGIL wasn't introduced to us until halfway through the semester, we didn't implement it exactly how it should have been. In an ideal POGIL situation, students would be reading the notebooks with themselves, and learning the material on their own, with TAs and the professor walking around to answer any questions. However, because the idea of POGIL was not introduced to the students, it was hard for them to get in the mentality of that kind of learning, and so the professor most of the time had to go through the worksheet with the whole class. If we start the class with groups, and introduce the idea of the POGIL worksheets early, we might have more success introducing this method of classroom interaction, as well as introduce the actual concepts in the course a lot more solidly, including the basics in the beginning.

4.3 Future Steps

As I mentioned before, as this project is only a senior thesis, the scope of my ability to work on this course ends once I graduate. [26] Therefore, I've listed a few of the ideas that people continue to research, for the future of this project.

For one, the final projects have not been looked at yet, as the due date for them was very close to the due date for this senior thesis. The projects should be looked at, and it should be decided how the analysis of what students did, and how complicated the projects were should be compared to the analysis in the rest of this paper, to have a full scope of how the class went.

Additionally, for future implementations of COSI 2A, the list of successes and future improvements should be looked at, and it should be decided what specific changes need to be made to try to replicate the successes of the course, as well as mitigate the challenges that came from the first implementation of the course. Many suggestions have been made, but the specifics of the improvements should come from whoever will be overseeing the course when it is next taught.

Finally, students who have taken the course this semester should be tracked through future computer science courses. For example, all though we asked in the final survey how many students from the course want to take another CS course, or pick up a major/minor, they should be tracked to see how many actually

[26] Though if more help is needed, I'd love to work on it part time :)

end up taking courses like COSI 11A or 12B[27], and how many actually pick up a major or minor. It will also be interesting to see if the demographics of students that end up trickling into these courses, to see if COSI 2A helped bring more diverse groups of people into the Computer Science major. Additionally, it would be interesting to see how they do compared to other students in the courses, to see if COSI 2A gave them a leg up in the course. All though the point of COSI 2A is to be a standalone course, it is definitely a benefit to get people so interested in programming that they wish to continue the major, and have them do better than people with no background in Computer Science. If this is true, COSI 2A could be used as a course for people who wish to be Computer Science majors but are hesitant to go into COSI 11A, as they heard it is too difficult. All though working on this course has been a lot of work, there is still definitely much research to do, much data to analyze, and lots of room for future research and projects into this type of course fitting into Brandeis University's culture.

5 Related Work

5.1 Computer Science Courses for Non Majors

Of course, the idea of a course in Computer Science for non-computer science majors is not a new concept. The New York Times article "Computer Science for the Rest of Us" [2] describes several universities with the idea of a course to teach computer science and computational thinking skills to students not intending on majoring in the subject. However, despite much research, we have not been able to find a course that has the same goals as COSI 2A. The article first cites Carnegie Mellon's "Principles of Computation" [28], which seeks to teach students about computation, from the history of the field, to basic and advanced programming, and then into the theory of computation, giving students an overview of the types of things they would expect to see in various computer science courses. In addition, while the NYT article states that the course uses Ruby, they have switched to Python. However, while the course gives a good understanding of theoretical computer science, it does not provide any understanding of how to use that programming outside of the course, and into the real life. Additionally, the rigorous nature of the course would have made it hard to attract the types of students that don't already want to do a Computer Science major or minor at Brandeis, as there isn't much new offered, and the goal of this course was to bring new people into programming. A course with a similar idea to this one, Harvey Mudd's "CS for All" [29] was a much friendlier introduction to the topics discussed in the CMU course, with open source texts discussing programming, functions, how computers work, and algorithmic "hardness." We ended up using quite a few of their readings for COSI 2A because of the friendly introduction to all this material. However, their dive into certain topics comes at the expense of learning about the real life applications that we wanted to push.

Some courses, such as UC Berkeley's CS10, try to introduce programming in Scratch, a graphical programming language made to visualize programming and avoid any form of syntax errors. While this is a good way to teach basic computational thinking skills, as it is very visual, it is hard to convince people that Scratch is anything like programming in the real world, as we believe it would scare students off programming languages even more. However, in the future, it might be interesting to start off with Scratch, and talk about computational thinking in those terms before moving to Python.

[27] Advanced Programming Techniques - the followup to COSI 11A
[28] Course found here: `https://www.cs.cmu.edu/~15110/`
[29] https://www.cs.hmc.edu/csforall/

Another course listed in that article was "Computing for Poets" at Wheaton College [30]. This course had a very interesting goal - to give English students the resources needed to use Python programming in their academic research. This trend in using programming in the humanities is referred to as Digital Humanities, which Wikipedia defines as "an area of scholarly activity at the intersection of computing or digital technologies and the disciplines of the humanities." [3] This is definitely a skill we wanted COSI 2A students to be able to pick up, and much of our course was based on the idea of this course. For example, this course also starts off with an introduction to programming, before talking about the applications in the digital humanities. The course also culminates in a final team project decided by the students themselves. Finally, the idea of the course making it easy for students who otherwise wouldn't think about programming to pick it up as a skill and use it to extend their major was a large part of what we wanted to accomplish with COSI 2A. Of all their students, less than 5% saw programming as any harder than learning a foreign language, and the course in general provided a safe space to be a novice. Interesting programs included one that would find a word that was used once in an author's entire corpus of text. All though the course material was way too specific to use for COSI 2A, many of the ideas were transfered onto this project, such as the idea of counting how many times a certain word appears in Romeo and Juliet when discussing reading files.

Finally, probably one of the most famous Introduction to Computer Science courses in the world, Harvard University's CS50 [31] is a course designed for people who intend on and who don't intend on starting a Computer Science degree at Harvard. According to their own FAQs, "CS50 is the College's introduction to computer science for concentrators and non-concentrators alike, with or without prior programming experience." The course is extremely successful both in terms of marketing and teaching students about all about how to use programming to their benefit. The course starts off teaching about Scratch,the language discussed above. From this, they go onto learning about arrays, algorithms, memory, and data structures using the C programming language. After this, the course introduces many other languages and ideas, including HTTP, machine learning, Python (with Flask and reading CSV files), SQL, and Javascript. By the end, the students are required to create a similar final project building off what they have learned during the course. This course has many of the things we were looking for in implementing COSI 2A, such as topics covered and focus on real life applications, and the final project that is up to the students. However, the course is known for how rigorous and fast paced it is, and as such attracts the types of students who are attracted to a course they will know will be difficult. We, however, wanted to do the opposite - attract the types of people who are hesitant to join a Computer Science course, and show them that getting started is not as difficult as it seems. Additionally, CS50 talks about an extreme range of programming languages and concepts, which we did not want to cover at the expense of not having the depth of understanding of one programming language.

In general, while all of the courses that are offered at other universities each had their advantages, we needed something that fit in the context of the culture at Brandeis, and the group that we're trying to reach, with the research questions we have in mind. We therefore needed to create our own course, instead of borrowing the model from somewhere else.

[30] Course website and syllabus here: `http://wheatoncollege.edu/lexomics/computing-poets/`

[31] https://cs50.harvard.edu/

5.2 POGIL and Jupyter Notebooks

The idea for using Jupyter notebooks came out of reading about POGIL, Process Oriented Guided Inquiry Learning. The idea is talked about in the Jupyter Notebook section, but it is basically about creating a course where the students learn how to learn for themselves - by discovering the tools they need to think critically about their subject material. The students are given worksheets with directed questions to guide their understanding of the lesson, as opposed to having a professor lecture at them. [1, 4] Using these worksheets was first intended as for chemistry courses, however, using them in Computer Science courses to increase retention is not new. [5]. However - what is new to our course is teaching the skills for using built in/external libraries, and using the POGIL methodology to create these skills. Additionally, while using Jupyter notebooks in this type of course is not a new idea [6], using Jupyter notebooks to facilitate on-computer POGIL learning seems to be something nobody has done before, and putting them together was the novel part of this thesis.

6 Conclusion

The purpose of this project was to create a course that was complimentary to the normal introduction to programming course at Brandeis University, for students who were interested in learning practical programming. All though there were some rough spots in the implementation, considering this is the first year it's been offered and planned, the students were extremely receptive to learning in this manner. The survey results indicate that many students were pleased with the course, with over 70% of them saying they would recommend the course to their friend if it was offered in the future. This type of feedback makes it very apparent that this type of course is one that needs to be at Brandeis University, as it fills a need for students of all majors that need programming competency to keep up with research and job prospects in their field. Additionally, the recruitment efforts from the previous semester were extremely valuable in getting a diverse group of people into the course, and having an extremely low drop rate through the semester. [32] Additionally, I believe that the idea of using Jupyter notebooks for POGIL style learning definitely has a future, as the format of the Jupyter notebook fits extremely well with the idea of POGIL worksheets, groupwork, and individual learning. I hope to see the course offered again, with a group effort from the Computer Science department and other departments, and have the next round follow the success of this course, with even more success to follow.

7 Contributions

My individual contributions to the COSI 2A project include:

- Coming up with the idea of the Introduction to Real Life programming, and coordinating with Professor Hickey to teach it.
- Exploring different types of Computer Science courses taught for non-majors, and how they approached the material.
- Creating and analyzing the surveys for COSI 11A and COSI 2A to analyze the reception of the course.

[32] Only 11/168 students dropped, with an even split on gender, and only one person of color.

- Exploring a wide variety of recruitment techniques and using surveys to follow up on their effectiveness, especially in getting a diverse group of people to take the course.
- Working with Professor Hickey to develop the learning objectives for both units of the course, and then following up on the effectiveness of those objectives through surveys.
- Researching APIs and external libraries to teach to form the second unit of the course.
- Developing Jupyter notebooks to teach about core python as well as about using external libraries and APIs (found in the Appendix).
- Created slideshow to introduce creating websites in Flask
- Acted as head teaching assistant for the course and attending all of the lectures, to have a first hand experience of what the course was like for the students.

References

[1] R. S. Moog and J. N. Spencer, "POGIL: An Overview," in *Process Oriented Guided Inquiry Learning (POGIL)*, vol. 994 of *ACS Symposium Series*, pp. 1–13, American Chemical Society, Sept. 2008. DOI: 10.1021/bk-2008-0994.ch001 DOI: 10.1021/bk-2008-0994.ch001 DOI: 10.1021/bk-2008-0994.ch001.

[2] R. Stross, "Computer Science for Non-Majors Takes Many Forms," *The New York Times*, Mar. 2012.

[3] "Digital humanities," Apr. 2017. Page Version ID: 775621084.

[4] S. L. Eddy, M. Converse, and M. P. Wenderoth, "PORTAAL: A Classroom Observation Tool Assessing Evidence-Based Teaching Practices for Active Learning in Large Science, Technology, Engineering, and Mathematics Classes," *CBE-Life Sciences Education*, vol. 14, p. ar23, Sept. 2015.

[5] C. Kussmaul, "Process Oriented Guided Inquiry Learning (POGIL) for Computer Science," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, (New York, NY, USA), pp. 373–378, ACM, 2012.

[6] A. A. Smith, "Teaching Computer Science to Biologists and Chemists, Using Jupyter Notebooks: Tutorial Presentation," *J. Comput. Sci. Coll.*, vol. 32, pp. 126–128, Oct. 2016.

[7] C. Holdgraf, A. Culich, A. Rokem, F. Deniz, M. Alegro, and D. Ushizima, "Portable learning environments for hands-on computational instruction: Using container- and cloud-based technology to teach data science," *arXiv:1703.04900 [cs]*, Mar. 2017. arXiv: 1703.04900.

[8] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonnier, "The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication.," *AGU Fall Meeting Abstracts*, vol. 44, Dec. 2014.

Appendices

A Jupyter Notebook For Strings/Methods

This tutorial will teach you the following things:

1. How to use built in string methods
2. How to learn about discovering and learning about methods
3. How to read online documentation
4. How to read from and write from files

As a rule to make this much more effective,

A.1 Part 1: Review

We already know what strings are, but just for a review: A string is a list of characters, which are like a word in Python. They let you store text.

```
x = "Hello!"
print(x)
```

We can also get strings from user input

```
name = input("What is your name: ")
print(name)
```

You can also add strings together using the + sign

```
print("Hello "+name)
```

You can get specific indexes of strings by using the indexing operations

```
print(x[1]) # Item at index 1 (the second item)
print(x[1:4]) # from index 1 to index 4
```

You can also use the str() function to turn something else into a string. This is very useful when adding other things to other strings.

```
num = 99
s = str(num)
print(s)
print(s + " Bottles of milk on the wall")
```

To get the length of the string, you can use the len() function

```
print(len(s))
print(len(x))
```

To go through each character in a string, you can loop through the characters in a for loop

```
def count_a(s):
    """
    This function counts the number of a's in a string s
    count_a("hello") = 0
    count_a("arya") = 2
    """
    counter = 0
    for character in s:
        if character == "a":
            counter = counter + 1
    return counter

print(count_a("arya"))
```

A.1.1 Project 1:

Write a function that counts the number of vowels in a string. Count the number of vowels in the string g (which has been created for you, don't touch that line), and submit it on TeachBack. For this situation, y is never a vowel.

HINT: It's very similar to the above code, but you need to change the if statement.

```
def count_vowels(s):
    pass # replace this with your code


# Once you finish writing count_vowels, the following code should give you the number of vowels in
    Romeo and Juliet.
# It's hard to test, but you can guess that it will be a lot of vowels.
g = open("romeo_and_juliet.txt").read() # open the file romeo_and_juliet.txt and turn it into a
    giant string
print(count_vowels(g))
```

A.2 Part 2: Built in Methods

Strings have a method called count. Here are a few examples of what it does, and how to call it.

What do you think it does?

Submit on Teachback what you think count does.

Here are a few examples:

```
s = "hello World"

x = s.count("o")
print(x)
```

Write a function using the "count" method on strings to return the number of a's in a string. This should be much shorter than the one given to you previously.

```
def count_as_better(s):
    pass # Replace this "pass" with your code. Use s.count() to count the number of a's and return
        that number
```

Here is another example of a method on s. It is called lower(), and it turns the string into a lower case version of itself. Type on teachback what you might think could be a good use for this method. Discuss with your group.

```
s.lower() # Turns the string into a lower case version of itself
```

What are some other methods that strings might have? Try a few by doing s.whateveryouthink(). Here is another example, but try to find your own. Don't google it yet, just try what you think might be a method on a string.

```
# Write some test code here.
```

Did you find any other methods? Type any you found on Teachback

A.2.1 How to find methods!

Obviously, programmers don't just guess what methods are named. It's a good skill to know though, how to guess what the thing you want to do is called, because you're often right. Remember that programming languages are made by people too :)

However, for an object (such as a string), there is a way to get all of the methods on that object! Simply call the dir() function on the object, like below. Ignore the functions with __'s around them

```
x = "Hello"
dir(x)
```

Try messing with a few of these functions, and figuring out what they do. Submit on TeachBack the ones you figured out what they do. If you get an error about the number of arguments, or the type of arguments, try messing with the types of arguments.

```
# Here is a cell for messing with code
```

A.2.2 AFTER YOU HAVE TRIED MESSING WITH CODE

Again, or course, there is a better way to find out what a function does than simply mess with it. There is another function, called help(), which will return the docstring for the function you pass in. For example, if we want to know what s.zfill does:

```
help(s.zfill)
```

Now we know we can try something like this:

```
x.zfill(10) # Pads x with some extra zeroes
```

A.3 Project 2:

Mess around with the s.islower() method. First, what do you think it does? (Hint: It doesn't take any parameters) - Try it on different strings, and then submit on TeachBack what you think it does.

Then - run the help function on s.islower, and see what the documentation says. Finally - submit on teachback one string that would make s.islower() True, and one that makes s.islower() False.

A.4 Another way to learn about methods

Using the dir() and help() functions is useful when you are just messing around in the terminal, but there's a better way to see all of the methods that a string has.

The Python documentation has a section of their website with descriptions of these methods: https://docs.python.org/3.6/l methods

A.5 Project 3

Go on it, and then try to answer these questions using methods on strings after reading through the possible methods you can use. The first one has been done for you.

```
s = "hello World"
```

```
# Write the code to turn s into HELLO wORLD (swapped case)
print(s.swapcase())
```

```
# Write the code to turn s into HELLO WORLD
```

```
# Write the code to turn s into hello world
```

```
# Write the code to turn s into Hello world
```

Another way to find out how to do something is to straight up Google it. For example, try to turn s into Hello World (every first letter capitalized)

First try to find the method, but if you can't, click this link. Then, try to find a good explanation of how to do the thing you want to do.

```
# Write the code to turn s into Hello World
```

B Jupyter Notebook on Reading and Writing Files in Python

This notebook will teach you the skills necessary to read and write text files.

B.1 Introduction

On your computer, you have probably opened a .txt file at some point. A txt file is just a file that has plain text, just like a string.

They are a common way of storing information on computers, and it is very easy to open and read from them.

For example, in the same folder as this notebook, there should be a file called romeo_and_juliet.txt

If you open this .txt file, you should see the entirety of romeo_and_juliet in your text editor.

Let's start with something simple. There's another text file called Grades.txt

The text file is just a list of people's names, with their grade after their name.

If you wanted to open it in Python to do some analysis on it, here's how you would do it.

```
f = open("Grades.txt") # First, open the file - you can save the opened file
```

```
f.read() # Then, you have you do f.read() to get the string version of the text file.
```

It would be worth it to save the f.read() into a variable.

```
f = open("Grades.txt") # First, open the file - you can save the opened file
s = f.read()
print(s)
```

Notice that when I printed this s, those weird
n's didn't show up. In a string,
n is the code that means new line, and when you give it to print, it will print out a new line. Look at this for example:

```
print("Hello\nMy\nName\nIs\nArya")
```

For files, the thing you probably want to do first is use the "splitlines" method on strings. This will split your string into a list of all the lines:

```
split = s.splitlines()
print(split)
```

```
# We can put all these three lines together into one line
all_the_lines = open("Grades.txt").read().splitlines()
print(all_the_lines)
```

Now we can do some data analysis on it. Let's first do a list comprehension to turn this into a lists of lists.

```
grades = [x.split() for x in all_the_lines] # .split() splits the string on the space, so now this
    is a list of lists.
grades
```

Now this is much easier! Now let's see what the average grade is.

```
# first let's create a list of just the score themselves, without the names,
# and lets convert the scores from strings to Python integers
scores = [int(line[1]) for line in grades] # why do we need to use the int(...) function??
print('the quiz scores are',scores)
print() # this prints a blank line!

# now that we have a list of the scores, we can easily find the average by dividing the sum by the
    length of the lists
avg_score = sum(scores)/len(scores)
print('the sum of the scores is',sum(scores),'the number of scores is',len(scores),'the average
    score is',avg_score)
```

Looks like we have a solid C average. I think we'll do better after taking the makeups!!

B.2 Files1

Add a few lines to the Grades.txt file using a code editor then write some code to read the file and find the highest grade!

B.3 Digital Humanities

Now let's look at some examples of working with Romeo and Juliet, which we downloaded from the www.gutenberg.org website.

```
# Let's find every line in Romeo and Juliet with the word "dog"

rj_lines = open("romeo_and_juliet.txt").read().splitlines()
# In one line, this opens the file, turns it into a string, and splits it into lines

# next we filter the list of lines to find only those containing the word 'dog'
result_lines = [line for line in rj_lines if 'dog' in line] # note the use of a list comprehension!

# now we print out the results
print('there are',len(result_lines),'lines in Romeo and Juliet containing the word dog',)
print('\nHere they are:\n')
for line in result_lines:
    print(line)
```

B.4 Line Numbers!

Here is a modification that finds the line numbers of the lines in Romeo and Juliet containing the word "dog". Observe that we create a list of the indices k of lines rj_lines[k] which contain the word "dog" and we can use those indices to print out the original lines...

```
# Let's find every line in Romeo and Juliet with the word "dog"
```

```
rj_lines = open("romeo_and_juliet.txt").read().splitlines() # In one line - opens file, turns it
    into string, and splits lines

# next we find out how many lines are in the Romeo and Juliet manuscript
num_lines = len(rj_lines) # let's see how many indexes we need to use

# here we find the line numbers (indexes) of the lines which contain the word 'dog'
# notice that range(0,num_lines) is the list of all line numbers for Romeo and Juliet

result_lines = [k for k in range(0,num_lines) if 'dog' in rj_lines[k]] # note the use of a list
    comprehension to filter

print('there are',len(result_lines),'lines in Romeo and Juliet containing the word "dog"')

print('\nHere they are:\n')
for k in result_lines:
    print(k,rj_lines[k])
```

B.4.1 Files 2: Modify the code in the previous example to also print the line before and the line after each occurrence of the word "dog".

So is should print

```
there are 6 lines in Romeo and Juliet containing the word "dog"

Here they are:

36 SAMPSON
37 A dog of the house of Montague moves me.
38 GREGORY

41 SAMPSON
42 A dog of that house shall move me to stand: I will
43 take the wall of any man or maid of Montague's.

....
```

with three consecutive lines around each occurrence of the word.
Cut/paste your code into TeachBack, as usual!

```

print('there are',len(result_lines),'lines in Romeo and Juliet containing the word "dog"')

print('\nHere they are:\n')
for k in result_lines:
    print(k,rj_lines[k])
```

B.5 How to store information in files.

Let's say you want to send some results to your boss in a text file. It's very easy to write things to text files. Here's how you do it.

```
f = open("newfile.txt", "w")
```

Notice that second argument "w" - This is how you tell python that this file doesn't exist, and you're going to create it, and "write" things into the file ("w" for write!)

Now to write a line to it, you can just write to the file like this:

```
f.write("Hello\n") # Don't forget the /n, that's how you make there be a new line in the text file.
f.write("World\n")
```

You can also "print" to a file by passing in the file like this as a second argument, file=f (where f is the name of your file). This is an example of "overriding a default parameter." You can define your own functions to have parameters with default values and allowing the user to override them. We'll talk about this later.

Notice that you don't need the
n here, since print adds it automatically.

```
print("Hello", file=f)
```

```
# Let's print a table of square roots
import math
for i in range(10):
    print(i,math.sqrt(i), file=f)
```

When you're done with the file you're writing, you have to close it. Like this:

```
f.close()
```

Now let's see the file! You can also open it in Atom.

```
print(open("newfile.txt").read())
```

C Jupyter Notebook for How to Use and Manipulate CSV Files in Python

C.1 Part 1 - What is a CSV file

A CSV file is a type of file where the data is structured into rows and columns, using commas and new lines. They're commonly used to represent data in a spreadsheet - such as from Microsoft Excel or Google Sheets.

For example, if I had this spreadsheet on Google Sheets or Excel

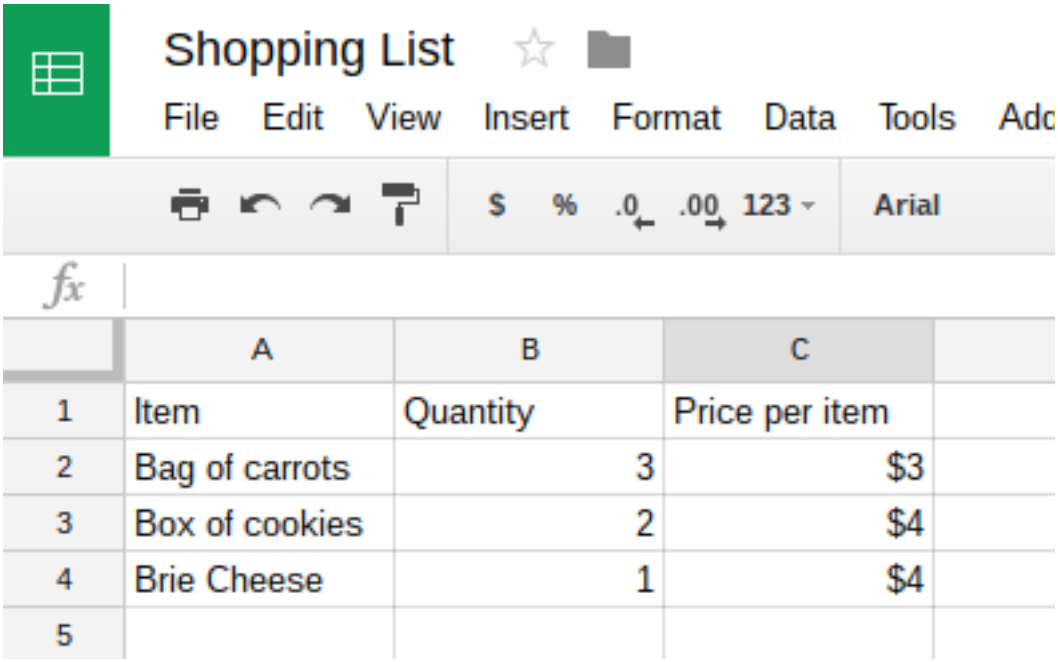


Figure 17: Sample Spreadsheet to convert to CSV

And then I went to File->Download As->Comma-separated values and downloaded the file, it would look like this: Item,Quantity,Price per item (in dollars) Bag of carrots,3,3 Box of cookies,2,4 Brie Cheese,1,4 In this example, the first row has three things separated by commas. For each of the following rows, the first item correspons to the first item in the first row (so Box of cookies is an item, 2 is a quantity, and 4 is a price per item.

P.S. If you can't remember which are rows and which are columns, you can think of columns like the roman columns (going up and down), and rows like running (left to right).

This shows probably the biggest reason we care about CSV files. They're structured, so they're easy to read programmatically, but they're also easy to just give to someone who knows nothing about code, so they can just open it in their favorite spreadsheet program. Many times when downloading large data sets from online, they will give you CSV files to read.

C.2 Part 2 - Reading CSV Files

C.2.1 Part 2a - Reading them manually

Remember that CSV files are just regular files with a standard formatting. Therefore, you can just read them like a regular file, and get the data you want.

For example, if we want to figure out how much we will have to pay in the end, we have to go through the rows, and multiply the price by the quantity, and add them all up.

```
csv_text = open("shopping_list.csv").read() # How to turn a file into a string
print (csv_text)
```

```
Item,Quantity,Price per item in Dollars
```

```
Bag of carrots,3,3
Box of cookies,2,4
Brie Cheese,1,4
```

```
# Now we have to split it on the new lines, so each
csv_text_split = csv_text.split(",")
print(csv_text_split)
```

```
['Item', 'Quantity', 'Price per item in Dollars\nBag of carrots', '3', '3\nBox of cookies', '2',
    '4\nBrie Cheese', '1', '4']
```

```
# We can get rid of the first line, since that's just the headers.
# We know that quantity is index 1, and price is index 2.
csv_text_split.pop(0) # Delete the line at index 0 - since it's the header
```

```
# And now we just iterate through each line, and take out the information we want.
total_price = 0
for line in csv_text_split: # For each line
    new_line = line.split(",") # split the line on the comma, so the line is now a list
        [Item,Quantity,Price]
    quantity = int(new_line[1]) # turn the thing at index 1 into an int
    price_per_item = int(new_line[2]) # take the dollar sign off the thing at index 2, and turn it
        into an int

    total_price += quantity*price_per_item

print("Total Price: ${}".format(total_price))
```

C.2.2 Part 2b - Using the CSV library

That wasn't awful - but there's a lot of code in there that would be repeated in all CSVs. In addition, our code doesn't handle some special cases (What if there are commas in the item, for example?)

Because of this, Python comes with a CSV library that makes it extremely easy to turn a CSV file into a list of lists, so that you can parse it more easily. Let me show you how it works:

```
# First step: import the library
import csv

# Second step: pass the open file into csv.reader
csv_lists = csv.reader(open("shopping_list.csv"))

# Third step: iterate through the file you created:
for line in csv_lists:
    print(line)
```

C.3 CSV1

Add a few lines ot the shopping_list.csv file using a code editor (e.g. atom) or a spreadsheet program (e.g. excel, but then store it as a CSV). Then execute the code above to print out the lines of data from the file.

We can find the total cost by looking at each line and multiplying the 2nd and 3rd values, quantity times price.

```
csv_lists = csv.reader(open("shopping_list.csv"))
total_cost = 0
next(csv_lists) # this advances the csv_lists by one, skipping over the header line

costs = [int(line[2])* int(line[1]) for line in csv_lists] # why do we need the int(...)'s
print(costs)

print('the total cost is $',sum(costs))
```

Something to keep in mind is that csv.reader isn't exactly a list of lists. It just goes through each line, and then becomes empty. So you can't read from csv_lists twice - the second time it will just be empty. This also means you can't do indexing on it.

```
print("here are the items in the csv_lists variable")
for line in csv_lists:
    print(line)
```

The reason the csv library does this is in case you had a very large CSV file - this way, you don't have to store it all in memory, you can just read it line by line.

To read from it more than once, you can convert it into a list after reading it. This will store the entire list in your computer's memory, and allow you to use it like a list of lists.

```
csv_lists = list(csv.reader(open("shopping_list.csv")))
print(csv_lists)

print(csv_lists[1][2])
```

With it as a list, we can also use a List Comprehension to get the total price.

```
csv_lists = csv_lists[1:] # Take off the headers
```

```
totals = [int(line[1])*int(line[2]) for line in csv_lists]
print(totals)
print(sum(totals))
```

Now you have a list of lists, which is the data from your CSV.

Try writing code that goes through csv_lists, and prints out the item you're spending the most money on.

C.4 CSV2

Write a function which finds the total number of items you are buying. Use a list comprehension to get the list of quantities of each item, then sum it.

```
def total_num_of_items(csv_lists):
    pass

csv_lists = list(csv.reader(open("shopping_list.csv")))
print(total_num_of_items(csv_lists))
```

You might have noticed a lot of annoying things about working with this library while working with it. For one, you have to drop the first row, since it doesn't contain any data you want. Secondly, you have to refer to the items by index, which means you have to know the index of what you want.

These issues can be solved with the DictReader module of the csv library. Let me show you how that one works, and what it produces:

```
# csv library is already imported
csv_file = csv.DictReader(open("shopping_list.csv"))

# now let's see what's inside
for line in csv_file:
    print(line)
```

As you can see, the DictReader takes in a CSV file, and gives you a bunch of dictionaries, where the key is the header, and the value is the value at that line. This makes it easy to write very readable code, as you can use the name of the header to get what you want. For example, to rewrite the "total cost" code:

```
csv_file = csv.DictReader(open("shopping_list.csv"))
total_cost = 0
for line in csv_file:
    quantity = int(line['Quantity'])
    price = int(line['Price per item'][1:])
    total_cost += quantity*price
print("Total cost is: ${}".format(total_cost))
```

The list comprehension version looks like this

```
csv_file = csv.DictReader(open("shopping_list.csv"))
print(sum(
    [int(line['Quantity'])*int(line['Price per item'][1:]) for line in csv_file]
))
```

It's up to you which version you want to use - whatever you're more comfortable with and you think looks the best.

C.5 Part 3: Writing a CSV file

Like reading CSV files, we don't necessarily need the CSV library to create a CSV. However, It makes it a lot easier. In fact, I will only go over how to create one using the CSV library.

Let's say we want to create a CSV of the first 100 numbers, and their values at x^2, x^3, and sqrt(x) Just like there's a csv.reader and a csv.DictReader, there's also a csv.writer and a csv.DictWriter. I'll show both ways of using them.

This also shows the "with" method of opening a file.

```
import math
# First way, using CSV writer
with open("number_values.csv", "w") as new_csv: # we add the extra 'w' parameter for saying this
    file will be written to
    writer = csv.writer(new_csv)
    writer.writerow(["Number", "Number Squared", "Number Cubed", "Square Root of Number"])
    for i in range(1,101):
        writer.writerow([i, i**2, i**3, round(math.sqrt(i),2)])
# Notice there's no "close" statement
```

```
with open("number_values.csv") as f:
    print(f.read())
```

The other way to do this is to use a DictWriter - I'll show you how to do that below. Remember that the way the DictWriter worked was that each line was a dictionary mapping the header to its value at that line. The writer will work similarly, for each line, we will write a dictionary.

```
with open("number_values2.csv", "w") as new_csv:
    # we have to tell the writer what our top fields are
    writer = csv.DictWriter(new_csv, fieldnames=["num", "squared", "cubed", "sqrt"])
    writer.writeheader() # to write the header
    for i in range(1,101):
        writer.writerow({"num": i, "squared": i**2, "cubed": i**3, "sqrt": round(math.sqrt(i), 2)})
with open("number_values2.csv") as f:
    print(f.read())
```

As you see, they produce the same output. It's up to you which one you want to do, depending on the type of CSV file you're trying to read/write from.

C.6 Real Data

C.6.1 Real Estate data

Download this CSV file: http://samplecsvs.s3.amazonaws.com/Sacramentorealestatetransactions.csv

Found from here: https://support.spatialkey.com/spatialkey-sample-csv-data/

"The Sacramento real estate transactions file is a list of 985 real estate transactions in the Sacramento area reported over a five-day period, as reported by the Sacramento Bee. Note that this file has address level information that you can choose to geocode, or you can use the existing latitude/longitude in the file."

We've already downloaded it into the file RE.csv in this folder, but you can download it again if you want. You should open it in excel or googlesheets and look at the data as a spreadsheet.

Next we look at the data using Python, and observe that it has 12 columns of data.

```
csv_file = csv.DictReader(open('RE.csv'))
for row in csv_file:
    print(row)
```

Finish these functions - the parameter will be a string which is the file name. We solve the first one for you!

```
# What is the average price of house sold?
def average_price(csv_filename):
    csv_file = csv.DictReader(open(csv_filename))
    prices = [int(line['price']) for line in csv_file]
    return( sum(prices)/len(prices))

print('the average price is',average_price("RE.csv"))
```

```
# return a list of all house with at least the specified number of bedroom and under the specified
    price
def find_house(bedrooms,max_price):
    pass
```

```
# What is the most expensive house?
def price_of_most_expensive_house(csv_file):
    # first find the prices then take the max and return it
    pass # write your code here
```

```
# Which zipcodes have the most expensive house sold?
def addresses_of_most_expensive_houses(csv_file):
    # first find the price of the most expensive house (by calling previous function!)
    # then find the addresses of the houses that cost M using a list comprehension
    # then return that list
    pass # write your code here
```

```
# Which house had the highest ratio of square feet to price
# In other words: which house was the most expensive per square foot?
# Return the address of the house
def most_expensive_house_per_sq_ft(csv_file):
    pass # write your code here
    # hint: find the list of prices per square foot using a list comprehension
    # then find the max of those values
    # then using another list comprehension to find the address of the house with the highest
        cost/sqft
```

```
    # and return that
```

C.6.2 Number 2:

Download an interesting CSV file online, and write code to find an interesting fact about it!

Here is an example of somewhere you can get an interesting CSV file: https://catalog.data.gov/dataset?res_format=CSV

D Jupyter Notebook for Introduction to APIs

D.1 Part 1: What is an API

API stands for "Application program interface." In layman's terms, it's an interface that allows you to communicate with another app, and send/recieve data to it. This is useful when someone else already built the tool you need, and you just need to use their data.

The difference between an API and a library is that a library will compute everything on your computer, while an API will just send a request to another service, and then give you back an answer.

Fortunately, a lot of APIs have Python interfaces, so you can use them as if they were libraries!

D.2 Part 2: How to use a Python-ported API

Let's say that you find an API for something, and it has a Python package you can install! This is great news, as it makes the work of interracting with the API EXTREMELY easy.

For an example, I will use the Indico API. This is an API for sentiment analysis - so you give it some text, and it can tell you some things about the text (such as the mood of the text, the political leanings of the person who said it, etc...). To do this, it uses something called Machine Learning, which might be discussed later. However, the idea is that they have a bunch of samples with data, and then based on those samples, make a guess on what the sentiment of a new piece of text is.

D.2.1 Part 2A: Indico API

The first step to using this API is to sign up for their website. Some APIs make you pay, but thankfully indico gives you 10,000 free queries a month (which should be plenty)

Just go to their website: www.indico.io, click "Get Started," and then "Sign up as you go."

Now that you have an account, you can go to their documentation and look at all the features available to use! The documentation is available here: https://indico.io/docs

They have many python samples to get you started, but I will take you through a few as well.

```
# The first thing you want to do is import the library
import indicoio
```

```
---------------------------------------------------------------------------

ModuleNotFoundError                       Traceback (most recent call last)

```

```
<ipython-input-1-44e367a4bbb5> in <module>()
      1 # The first thing you want to do is import the library
----> 2 import indicoio


ModuleNotFoundError: No module named 'indicoio'
```

The next thing you want to do is add your config-id to the indicoio object. However, the way that the code sample does it is insecure, since you don't want anyone else to be able to see your key.

There's a way we can get around this though. In the same folder as your code using indico, make a file called secrets.py In secrets.py, you should only have one variable:

indico_key = "whatever_your_key_is"

You can get your key from the documentation, or from the top of your dashboard: https://indico.io/dashboard/. I've included a file "secrets_example.py" that you can look at, but make sure you change the name to secrets.py, and have the correct API key.

And then in your code, you can get the key like this.

```
from secrets import indico_key
indicoio.config.api_key = indico_key
```

And now you're all authenticated for using your indico account! Let's run some queries.

```
# Let's find the sentiment of a string
result = indicoio.sentiment_hq("Today was a very good day!")
```

```
print(result)
```

```
0.9157847166
```

This result is a bit confusing, so let's see what the documentation says:

> This function will return a number between 0 and 1. This number is a probability representing the likelihood that the analyzed text is positive or negative. Values greater than 0.5 indicate positive sentiment, while values less than 0.5 indicate negative sentiment.

So 0.91... indicates a result that has a very high probability of being positive. Let's try a few more, just to see how well it works.

```
indicoio.sentiment_hq("Our presentation was a disaster")
```

```
0.0328917056
```

```
indicoio.sentiment_hq("You have a nice computer")
```

```
0.9253799319
```

```
indicoio.sentiment_hq("I end work at 5pm")
```

```
0.5880327225
```

```
# We can also pass it a list, and it will return a list of results
indicoio.sentiment_hq(["How are you today?", "I'm doing great, thank you!"])
```

```
[0.7249644995000001, 0.9912720919]
```

Feel free to try a few more of these just to see. Some useful applications of this may include analyzing Yelp reviews of your restaurant, analyzing chats with a friend, or analyzing novels to plot out an emotional graph of how the novel turned out.

I'll show some examples of other API calls you can make. A lot of them return dictionaries, which contain information.

```
# First paragraph in article about Computer Science on wikipeida
paragraph = """Computer science is the study of the theory, experimentation, and engineering that
    form the basis for the design and use of computers. It is the scientific and practical
    approach to computation and its applications and the systematic study of the feasibility,
    structure, expression, and mechanization of the methodical procedures (or algorithms) that
    underlie the acquisition, representation, processing, storage, communication of, and access to
    information. An alternate, more succinct definition of computer science is the study of
    automating algorithmic processes that scale. A computer scientist specializes in the theory of
    computation and the design of computational systems.[1]

Its fields can be divided into a variety of theoretical and practical disciplines. Some fields,
    such as computational complexity theory (which explores the fundamental properties of
    computational and intractable problems), are highly abstract, while fields such as computer
    graphics emphasize real-world visual applications. Other fields still focus on challenges in
    implementing computation. For example, programming language theory considers various
    approaches to the description of computation, while the study of computer programming itself
    investigates various aspects of the use of programming language and complex systems.
    Human-computer interaction considers the challenges in making computers and computations
    useful, usable, and universally accessible to humans."""
```

```
# nobody has time to read all of that
result = indicoio.keywords(paragraph)
print(result)
```

```
{'computer': 0.09608448800000001, 'universally accessible': 0.1869326081, 'programming language
    theory': 0.0322087061, 'theory': 0.1262267688, 'computer graphics': 0.0517569035, 'basis':
    0.0499561376, 'complexity theory': 0.0850902368, 'complex systems': 0.0350902368, 'computer
    science': 0.1289636064, 'experimentation': 0.0618151058, 'programming language': 0.0643414067,
    'computer scientist': 0.1350902368, 'scientific': 0.10830214910000001, 'computer interaction':
```

```
    0.1501730263, 'computational complexity theory': 0.0322087061, 'computational complexity':
    0.1012881383, 'science': 0.108400676, 'study': 0.06473109860000001, 'practical approach':
    0.1388250084, 'computer programming': 0.0350902368, 'fields': 0.1012478037, 'theory of
    computation': 0.0665995761}
```

This gives us a dictionary of all the keywords in the paragraph, with the probabiity that they are important to the paragraph. We can pass it a parameter "top_n" which will limit it to only that many keywords

```
result = indicoio.keywords(paragraph, top_n=7)
print(result)
```

```
{'computer scientist': 0.1350902368, 'theory': 0.1262267688, 'science': 0.108400676, 'practical
    approach': 0.1388250084, 'computer science': 0.1289636064, 'universally accessible':
    0.1869326081, 'computer interaction': 0.1501730263}
```

The result is just a regular python dictionary, so we can loop through these the same way we would do it for any dictionary. This would also be interesting if passed in paragraphs from a book, see what the keywords are. Maybe make a graphic based on the probabilities

```
# Here is an example of the political analysis API
# obama_farewell.txt is the contents of Obama's farewell speech to the nation
speech = open("obama_farewell.txt", encoding="utf8").read()
result = indicoio.political(speech)
print(result)
```

```
{'Libertarian': 0.08965711300000001, 'Liberal': 0.7911154628, 'Green': 0.0136871245,
    'Conservative': 0.1055402532}
```

```
# Unsurprisingly it is 80% Liberal. You could break it down sentence by sentence, to see which
    parts were most liberal
# We can try the same with Donald Trump's victory speech
speech = open("donald_victory.txt", encoding="utf8").read()
result = indicoio.political(speech)
print(result)
```

```
{'Libertarian': 0.1327656806, 'Liberal': 0.3615416288, 'Green': 0.1044029593, 'Conservative':
    0.4012897909}
```

Somewhat surprisingly, this speech looks almost as liberal as it looks conservative. I'm not a political scientist, but using these kinds of APIs is a good way to find trends in large sections of data. For example, analyze all of Obama's speeches over time and see if Indico finds any trends, and look at the data and see if they're substantial in any way. The useful thing to note is that the computer has no biases towards certain people, so it will be better at finding trends because of that.

```
# Last one to try: Indico can analyze images
# https://indico.io/docs#image_recognition

# Let's see if it can tell which what's in this image
image_url =
    "https://www.sciencenewsforstudents.org/sites/default/files/2016/06/main/articles/860-header-dog-breeds.jpg"
indicoio.image_recognition(image_url, top_n=10)
```

```
{'Afghan hound, Afghan': 0.076512441,
 'Blenheim spaniel': 0.0315301642,
 'English foxhound': 0.0778390318,
 'Saint Bernard, St Bernard': 0.2607925236,
 'Walker hound, Walker foxhound': 0.0526631102,
 'Welsh springer spaniel': 0.025802087,
 'beagle': 0.0403615013,
 'dogsled, dog sled, dog sleigh': 0.15499596300000001,
 'papillon': 0.0283743348,
 'pug, pug-dog': 0.0303313788}
```

I don't know if these breeds are the real ones, but if they are then this is impressive! Try with your own images.

D.3 Exercises

D.3.1 Question 1

Use the personality API on Indico to find the most likely traits of Donald Trump and Barack Obama, given their speeches. Does the result surprise you?

```
def get_personality(text):
    pass
```

D.3.2 Question 2

Do something interesting with the API! It's up to you to find some corpus of text, and find out something interesting about it.