The Digital Aristotle

The New Mexico Supercomputing Challenge
Final Report

April 1, 2015

Team 137
Saint Pius X High School

Team Members: Thomas R. Curtin and Samuel J. Gervais
Mentor: Jordan E. Medlock

Correspondence: samgervais512@gmail.com

Running Title: The Digital Aristotle

TABLE OF CONTENTS

1. Abstract

Traditional education is generalized and inefficient in society and can be improved on. Traditional tutoring while better than traditional education, still contains significant flaws and can be perfected through the use of computing. More individual and accessible means of tutoring can be created to teach both students who are struggling and excelling the information best suited for them. This will be achieved by the use of searching capabilities and adaptive code to create a more individual and online tutor. Also lectures will be added to allow a more dynamic and less linear learning experience. For the first year, Digital Aristotle contains a database of open source knowledge and returns pertinent information based on the student's input.

2. Background

Throughout history, humans have searched for better ways to gain knowledge and share it with others. This desire to learn created the education system seeking to share knowledge with everyone. However, the education system has many flaws, most prominently in the lack of individualized attention given to students that are either struggling or excelling in the subject matter. This method used in schools is inefficient and needs a more dynamic and individual approach to ensure better understanding of the subject material. To fix this tutoring was created to fill the need of individualized attention to better develop a student's understanding of the subject.

Although tutoring is a step above normal education, there are still significant problems and room for improvement. These problems stem from the need of a human to identify the needs of the student and decide the best way for the student to learn. This process is limited in the lack of tutors to fill the need of the students, the expense of the tutor to tutor the student. These problems make traditional tutoring inefficient compared to other means of tutoring. Other tutoring programs have been created to more efficiently teach students.

Other online websites have been created to solve these problems with traditional tutoring, one being Kahn academy. Kahn academy takes a different approach to giving students the assistance necessary to understand the subject. They use tutoring videos and online assignments to help students understand curriculum to solve the problem of expense and amount of tutors. This method however good, still has flaws that have to be addressed. For one the individualization of tutoring is not as prevalent as the free aspect of Kahn academy. This is a problem because the way Kahn Academy teaches is not

effective to all students. This lack of individual attention is still covered by traditional tutoring which allows for improvement on both.

Another prominent tutoring system is Aleks. Aleks takes a different approach than Kahn Academy by focusing on the individual aspects of tutoring rather than the human aspects. Aleks uses adaptive questioning to determine what the student is most ready to learn. This process is flawed due to the lack of human interaction which is needed to round out the learning experience. This lack of interaction is also fulfilled by the still imperfect system of tutoring and can be expounded on with the use of all elements of tutoring. All tutoring systems seem to have a lack of well-rounded teaching ability which is prevalent in traditional tutoring which has room for improvement.

3. Problem Statement

We are creating the Digital Aristotle with the desired goal of bringing greater and more achievable education to students who face hardships with a certain subjects. This goal is essential due to the complex nature of material, especially in mathematics, which causes great confusion for the student. The way subject material is taught in a traditional classroom is set to be directed linearly and within a specific time range, therefore subject material is either taught too fast or too slow for the students. The gaps in understanding for students who do not comprehend the material at the rate of the classroom not only hurts their studies while they are learning the material, but it also hurts their future progress with that material; this is especially prominent mathematics because in all fields, advanced parts are based off of more basic parts. There are also not enough teachers in comparison to every student to be able to effectively give personal and individualized recognition towards the scholar's complications. This is an unavoidable consequence which forces the student to maintain large misconceptions that cannot be easily and quickly explained. Furthermore, we are making the Digital Aristotle to be used freely on a webpage in order to allow for more students to gain access to the information without the high costs of a human tutor. This gives most students the ability to receive individualized tutoring in a convenient manner because of the Digital Aristotle's form of a webpage. The webpage, with its main use of a search bar, also gives the user the capability to ask questions on material that they do not understand, and receive a guided answer to benefit their knowledge. Lastly, the Digital Aristotle is created to aid students who struggle in their studies and seek to find greater knowledge outside the classroom in assistance with their course.

4. Methods

A largely important aspect of the Digital Aristotle is its user-interface conveniently based in a webpage. The most prominent use of the user-interface is with the search bar displayed on the webpage in which the user can type text into, and the input is used to determine the output of pages displayed on the page. The client-side of the webpage is comprised of three documents containing its programming: HyperText Markup Language (HTML), JavaScript with JQuery, and Cascading Style Sheet (CSS) files. The HTML file loads the scripts for JavaScript, and CSS style which ultimately allows for the use of the webpage beyond only displaying text; it also creates the search bar and output which is used as the main user-interface device. The JavaScript document contains the main functions of the webpage as it is used to store the user's input, and use that input to produce images which most closely relate to the input. It does this by saving the inputted text as a string in a variable which is then slightly pruned in order to remove unnecessary spaces at the beginning and end of the string. After this, the variable is sent to a document-locating program which compares the input to multiple different texts in a database, and the names and file locations of the pictures that most close relate to the input are returned as a variable. Finally, these pictures are displayed in order on the webpage for the user to read and use. The CSS, on the other hand, contributes primarily to the aesthetics of the webpage as opposed to the JavaScript main use being for the actual function of the webpage. The CSS is used to align the text, search bar, output, and images on the page; it also adds margins and color to the background of the page. In short, the user control of the Digital Aristotle is used on a webpage which brings up information based upon the user's input into the search bar.

The server-side portion of the webpage creates the link between the human-computer interaction aspect of the program and the document-locating program. The server program mainly uses the Tornado web framework, an asynchronous web server based in Python which is used for quick and continuous use of the web server. The program creates multiple handling classes that are used to display the client-side portions to on the webpage, and displayed to the user; another class is created to receive the user's input in the search bar, and then display the results of that from the document-locating program. The handling classes first find the Multi-Purpose Internet Mail Extensions (MIME), a process used to identify files on through a server based off of their format, of a file that will be used with the webpage. This opens the file in the background as an object with Python's "open" function, and it is then created in a binary form using Python's "read" function. This can be used and displayed by the Tornado server for the user to utilize. The searching class works by creating the database from the document-locating program, and then sends the user's input to the server from the JavaScript document to find the results based off of the input from the document-locating program. Subsequently, the results are sent to be displayed on the webpage through the server. These classes are assembled using Tornado's request handler where the classes are applied with their corresponding files and components that they use to display and include on the webpage. The use of this server and webpage allow for the more customizable application for the Digital Aristotle, and the more efficient demonstration of the results. These elements create the main use and function for the human-computer interaction in order to make it as advantageous for the user as possible. In conclusion, the server uses numerous classes in order to display components of the webpage for the user, and also so that it is able to function

quickly and efficiently with the client-side portion as well as to produce the most efficient and utilitarian human-computer interaction.

The main facet of the Digital Aristotle is its knowledge database which is created from the set of free e-books provided by Project Gutenberg. The database creation is located within the document-locating program, and it is created as a JSON (JavaScript Object Notation) Schema. The database is composed in a cascading style where a book object contains a list a chapters which contains a list of subchapters which contains the text of the book. The main book object is the database encompasses the simplest set of keys and values, but it is necessary in order to easily access the more important chapters and subchapters object; the book object only contains the actual book's name, chapters, and, if available, the publish date and isbn. The chapters object are comprised of the chapter's name, subchapters in that chapter, and all of the pages within that chapter. The chapter's name is used by the document-locating program to test if the chapter is about what the user searches, and if it is, then the pages value is used by the webpage to display those pages. Then, the subchapters object contains the subchapter's name, the text in that subchapter, and the amount of pages in the subchapter. The name of the subchapter is used correspondingly to the name of the chapter, but the text is based accumulatively to the similar words in the text compared to the user's input. The database in this format is created in Python using a class object for each of the elements. The first class created is used by the other class objects so that they can be serialized which is the process that allows for the database to be written in JSON and used in Python. The next classes, the Book, Chapter, and SubChapter classes, are created with the contents mentioned previously, and each of them inherit the serializable class which allows each of them to later be used by the

document-locating Python functions. After all of these classes are defined, a function is used to actually make the database in the correct composition. The creation begins by making a variable that contains the Book object, and another that contains a JSON document based off of the Book which has each of the book's pages and the words with their fonts on that page. Each page is then searched, and based on the font of a group of words, they are categorized into chapters and subchapters. As a final step, the JSON file produced is modified so that it can be searched through and read by Python with the same categories as in the JSON database. Briefly, multiple classes are used to create a JSON database with book, chapters, and subchapters objects which can be used by the document-locating functions to return the correct chapter or subchapter based on its contents.

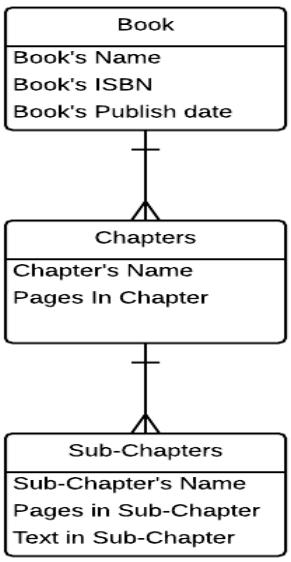

Figure 1. A representation of the JSON database format with its sub-sections.

The document-locating program is used to find the pages of a chapter or subchapter after comparing an input to the name and text in the chapter or subchapter. This process is completed using many functions, the most important of these being the levenshtein distance. The levenshtein distance is a method used for measuring the differences between two strings, and it is used to compare the differences between an input and the title of a chapter or subchapter in order to produce a fuzzy search. The levenshtein function compares the two strings by matching each of their characters and setting a different value based on the action needed to get them the match (i.e. insertion, substitution, or deletion). The main locating function is used to find an index which uses the levenshtein distance on the chapter and subchapter, and the text in the subchapter to find a more accurate comparison of the input and chapter or subchapter. This function first creates a nested function which is used to create a negative accumulator where every word that is not a stop-word, words that are not relevant to data search, but used in spoken and written English, matches the input subtracts one form this accumulator; the levenshtein distance of the word in the chapter or subchapter compared to the input divided by each of the words in the subchapter is then added to the accumulator. In order to make this process quicker, the process is “memoized” which is the technique of storing results of a function that use bulk amounts of processing power with repeated or redundant calculations. After this, another nested function is defined which creates an index that starts as being the levenshtein distance of the chapter or subchapter’s name. This index is modified differently depending on if the type of the chapter being used is a chapter or subchapter. If it is a subchapter, then the index is only modified by adding its accumulator found in the function described before. On the other hand, if it is a chapter, then another accumulator is created which contains the sum of the accumulators of each subchapter found by the other

nested function. The average accumulator is then found by dividing that by the total number of subchapters in the chapter, and that is added to index. This can be signified with the following function where *W* represents the average number of words the subchapters, *S* represents the number of subchapters present, and *L* represents the levenshtein distance of every word:

$$index = \frac{-\sum_{n=0}^{W(S)} n}{S} + L$$

A list of these indexes are created, and the another function is used to find the smallest of these by using Python's ability to create a variable representing infinity to compare each index to this variable until the chapter with the smallest index is found and returned. A final searching function is defined which uses these functions to compare the string of the name of a chapter or subchapter and the name's individual words, and return the chapter with the lowest index. To conclude, the levenshtein distance between an input and the chapter or subchapter's name and text is used to create a fuzzy search that is able to accurately find a matching chapter or subchapter.

In conclusion, the user control of the Digital Aristotle begins on a webpage where the user has the ability to input a string of characters; this input is then sent by the server to a document-locating program. The document-locating

program allots the text to numerous functions which mainly use the Levenshtein distance, a method used for measuring the differences between two strings, and compares the text to chapters, sub-chapters, and text held within the JSON database. The database is created by “searching” through a PDF of a mathematics book and using the disparity in font between sections in the book to construct a codified database in which a more accurate result will be produced based upon the user’s input. After the chapter which most relates to the input string is found, the chapter’s pages are displayed for the user on the webpage.

5. Results

The Digital Aristotle, being a four-year project, is far from its culmination, but the first year or stage of the project has been completed. In this stage, a database containing mathematics from free e-books provided by Project Gutenberg has been forged and culled. A document-locating program has also been created with the use of searching through this database, and an intricate use of functions have been made to compare an input to the many chapters and subchapters in the database in order to find the chapter or subchapter that best compares to that input. For simple use of the program, we have made a webpage with a simple search bar that is joined with the much more complicated document-locating program for the user to add their input, and the pages of the book relating to that input are displayed below for the user to employ. This webpage is connected to a customizable server which links the webpage to the document-locating program. These are what the first year of the project is comprised of, and it is all functional without many complications. A functional server and webpage have also been created which makes the program more accommodating, so the first-year of the project is operable.

6. Conclusion

Education is a key part of society that impacts every citizen looking to excel in life. Currently the approach to teaching students is flawed due to the generalized teaching in classrooms rather than individual attention. Tutoring tries to fill this lack of individualized attention by having a teacher teach the material in what they believe is the most effective to their students understanding. Tutoring still has flaws that can be improved on such as the amount of tutors compared to the amount of students. Other flaws include the expense of hiring a tutor and the time it takes to tutor. The use of adaptive code in Digital Aristotle creates an individualized attention for the students by understanding their strengths and weaknesses. Digital Aristotle impacts the foundation of traditional education by being an easy source to gain a deeper understanding of the knowledge sought. As the first year closes on this project, Digital Aristotle is a base foundation of what is going to be accomplished. The search engine allows for students to search for specific topics within the database and allows them to find relevant information to assist them. The full project solves the problem of students needing individual assistance by using a adaptive algorithm to adapt to the students strengths and weaknesses and to have a human element as well. This approach is unique to Digital Aristotle through the use of multiple tutoring methods used in cooperation.

7. Discussion

The Digital Aristotle tutoring system, much like its human namesake, seeks to transform selective areas of knowledge and effectively communicate this knowledge to students. Although the scope for development of Digital Aristotle in the first year of this multiyear project was limited by design, significant progress in planning continuous improvements in Digital Aristotle. The Digital Aristotle tutoring system consists of a database of knowledge commonly referred to as the knowledge-base, an overarching knowledge retrieval subsystem, and a human-computer interface enabling input queries and output visualization of retrieved information. The first year effort created a baseline system comprised of a knowledge-base and search engine that retrieves information from an algebra textbook based on user input queries. Completion and use of the baseline Digital Aristotle system revealed many insufficiencies. These insufficiencies guiding our planning for future improvements and upgrades to Digital Aristotle. Most notably we plan on increasing the content and diversity of the knowledge-base. In addition, the lack of selectivity in retrieved information, and lack of specificity pertaining to the user input query will need to be fixed as well. This project will be improved on throughout the next 4 years as it is an ambitious project. This organizational system will better segment the goals of Digital Aristotle.

8. Acknowledgements

We would like to thank Jordan E. Medlock for his invaluable knowledge in programming and his assistance throughout this entire project. We would also like to thank our parents for constant support and help editing this paper. Lastly we would like to thank CGP Grey for piquing our interest in the topic of individualized education.

9. Bibliography

Kahn, Salman. "Khan Academy." *Khan Academy*. N.p., n.d. Web. 29 Mar. 2015.

NYU, and UCI. "ALEKS -- Assessment and Learning, K-12, Higher Education, Automated Tutor, Math." *ALEKS -- Assessment and Learning, K-12, Higher Education, Automated Tutor, Math*. N.p., n.d. Web. 29 Mar. 2015.

Clarke, Wallace. *A First Book in Algebra*. N.p.: n.p., n.d. *Project Gutenberg*. 27 Aug. 2004. Web. 24 Mar. 2015.

"NumPy." *NumPy — Numpy*. N.p., n.d. Web. 31 Mar. 2015.

"Tornado." *Tornado Web Server — Tornado 4.1 Documentation*. N.p., n.d. Web. 31 Mar. 2015.

"SciPy.org." *SciPy.org — SciPy.org*. N.p., n.d. Web. 31 Mar. 2015.

10. Appendix

10.1. Software

Tornado - http://www.tornadoweb.org/en/stable/

Tornado is a Python web framework and asynchronous networking library.

PIL - http://www.pythonware.com/products/pil/

The Python Imaging Library (PIL) adds image processing capabilities to your Python interpreter. This library supports many file formats, and provides powerful image processing and graphics capabilities.

Levenshtein - http://www.levenshtein.net/

The Levenshtein distance calculates the differences between two strings to determine how to modify one string to obtain another string.

NumPy - http://www.numpy.org/

NumPy is the fundamental package for scientific computing with Python. It is a powerful N-dimensional array object, provides sophisticated (broadcasting) functions tools for integrating C/C++ and Fortran code, and useful linear algebra.

SciPy - http://www.scipy.org/

SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering. It works in cooperation with NumPy and Matplotlib.

Matplotlib - http://matplotlib.org/

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

NLTK - http://www.nltk.org/

The Natural Language Toolkit (NLTK) is needed for building Python programs to work with human language data.

10.2. Source Code

```
"""
Authors: Samuel J. Gervais and Thomas R. Curtin
School: Saint Pius X High School
Email: samgervais512@gmail.com
Description: Main database creation, search, and document-
locating program
"""
####### Modules Begin

import json
from matplotlib.pylab import *
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.colors as colors
import random
import time
from itertools import chain
import nltk
import os.path

####### Modules End

###### Data Components Begin

class Serializable(object):
    '''
```

```
    Creates a class that allow for the creation of a
    JSON database while still being able to be used in Python
    '''
    @classmethod
    def fromJsonFile(cls,fname):
        f = open("Math6")
        document = json.loads(f.read().encode('utf8'))
        return cls.fromJsonObj(document)

    @classmethod
    def fromJsonObj(cls,obj):
        pass

    def toJSONObj(self):
        pass

    def __str__(self):
        return (json.dumps(self.toJSONObj(), indent=2))

class Book(Serializable):
    '''
    Creates a class that will contain major components
    (book's name, chapters, isbn, and publish_date) of
    the database based on the math book
    '''
    #                 str  [Chapter] str         str
    def __init__(self,name,chapters,publish_date,isbn):
        self.name = name
        self.chapters = chapters
        self.publish_date = publish_date
        self.isbn = isbn

    @classmethod
    def fromJsonObj(cls,dictionary):
        '''
        Creates a dictionalry with components of the book
        '''
```

```
        return cls(dictionary['name'],[Chapter.fromJsonObj(obj)
for obj in
dictionary['chapters']],dictionary['publish_date'],dictionary['i
sbn'])

    def toJSONObj(self):
        '''
        Adds information to dictionary based on dictionary
        values created in the "fromJSONObj" function
        '''
        return {'name' : self.name, 'chapters' :
[obj.toJSONObj() for obj in self.chapters], 'publish_date':
self.publish_date, 'isbn': self.isbn}

class Chapter(Serializable):
    '''
    Makes a class which will contain the a chapter's name,
    SubChapter object, and the number of pages in the entire
chapter
    '''
    #                 str  [Sub_Chapter]
    def __init__(self,name,subchapters,pages):
        self.name = name
        self.subchapters = subchapters
        self.pages = set(pages)

    '''
    Next two functions do the same JSON creation as in the
"Book"
    class, except it uses dictionary values for the chapter's
name.
    subchter, and pages
    '''
    def toJSONObj(self):
        return {'name' : self.name, 'subchapters' :
[x.toJSONObj() for x in self.subchapters],
'pages':list(self.pages)}
```

```
    @classmethod
    def fromJsonObj(cls,dictionary):
        return
cls(dictionary['name'],[SubChapter.fromJsonObj(obj) for obj in
dictionary['subchapters']],dictionary['pages'])

class SubChapter(Serializable):
    '''
    Makes a class which will contain a subchapter's name, the
    text in the subchapter, and the amount of pages in the
SubChapter
    '''
    #                   str [Paragraph]
    def __init__(self,name,text,pages):
        self.name = name
        self.text = text
        self.pages = set(pages)

    '''
    The next two fucntions do the same JSON database creation as
in the
    "Book" class, except it uses dictionary values  for the
SubChapter's
    name, text in the SubChapter, and the number of pages in the
subchpter
    '''
    @classmethod
    def fromJsonObj(cls,dictionary):
        return cls(dictionary['name'],
                   dictionary['text'],
                   dictionary['pages'])

    def toJSONObj(self):
        return {'name' : self.name, 'text' : self.text, 'pages':
list(self.pages)}

###### Data Components End
```

```
####### Data Creation Begin

def createNewMathOut():
    '''
    Function to use classes in "Data components" to create the
JSON database
    by searching through each page of the book, and using the
font to determine
    where the type of object, chapter, subchapters, or text, the
information belongs
    '''
    f = open("LinedMath.json")
    book = Book('A First Book in Algebra',[],"","")
    document = json.loads(f.read().encode('utf8'))
    chapter = None
    subchapter = None
    for (page_num, page) in enumerate(document):
        page_num += 1
        for line in page: # Searches by each line of text on a
single page
            if line[u'font'] == 5: # chapter
                chapter = Chapter(line[u'data'], [], [page_num])
                book.chapters.append(chapter)
                subchapter = None

            elif line[u'font'] == 3: # Subchapter
                subchapter = SubChapter(line[u'data'], '',
[page_num])
                if chapter is None:
                    chapter = Chapter('No Name Chapter', [],
[page_num])
                chapter.subchapters.append(subchapter)

            elif subchapter is not None:
                subchapter.text += u'\n' + line[u'data']
                subchapter.pages.add(page_num)
                chapter.pages.add(page_num)
```

```
            else:
                subchapter = SubChapter("No Name",
line[u'data'], [page_num])
                if chapter is None:
                    chapter = Chapter('No Name Chapter', [],
[page_num])
                chapter.subchapters.append(subchapter)

    fout = open("newMathOut.json",'w')
    string = json.dumps(book.toJSONObj())
    fout.write(string) # Creates the database as a text file

def readBook():
    '''
    Function used to search through the JSON in python
    database created in the fucntions
    '''
    f = open('newMathOut.json')
    book = Book.fromJsonObj(json.loads(f.read()))
    return book

####### Data Creation End

####### Search Data Begin (Functions)

def levenshtein(source, target):
    '''
    Function used to compare two strings for their similarity by
comparing
    the individual characters in the strings, and finding what
is needed to
    make them the same (i.e. insertion, substitution, or
deletion). Returns
    a number based on the changes required; the smaller the
number, the closer
    the two strings are to eachother.
    '''
    if len(source) < len(target):
```

```
        return levenshtein(target, source)

    if len(target) == 0:
        return len(source)
    source = np.array(tuple(source))
    target = np.array(tuple(target))

    previous_row = np.arange(target.size + 1)
    for s in source:
        # Insertion
        current_row = previous_row + (0.1)

        # Substitution or matching:
        current_row[1:] =
np.minimum(current_row[1:],np.add(previous_row[:-1], (target !=
s) * (3)))

        # Deletion
        current_row[1:] = np.minimum(
                current_row[1:],
                current_row[0:-1] + (1))
        previous_row = current_row

    return previous_row[-1]
levenshtein = np.vectorize(levenshtein)

def minimumChapter(xs):
    '''
    Function used to find the chapter with the least amount of
difference
    between the input and a (sub)chpter's text and name given an
index where
    the smallest index is the (sub)chapter with the least
difference
    '''
    smallest = float("inf")
    smallestChapter = None
    for (value, chapter) in xs:
```

```
        if value < smallest:
            smallest = value
            smallestChapter = chapter
    return smallestChapter

numberOfTimesRan = 0
filtered_words_global = []
def bestChapter(input_string, chapters):
    '''
    Function used to return the chapter or subchapter withe the
smallest
    difference between the input. It compares all of the
chapters and
    subchapters to the input
    '''
    def countWordsInSubChapter(subchapter):

        '''
        Function that changes a subchapters text into a lost of
individual
        words, and then removest irrelevant words, stop words,
from this list.
        It then creates a negative accumulator where one is
subtracted from it
        if the word matches the input. The sum of levenshtein
distance of a word
        divided by every word is then added to this accumulator.
        '''
        global numberOfTimesRan, filtered_words_global
        numberOfTimesRan += 1
        accum = 0
        text = subchapter.text
        words = text.split()
        filtered_words = [w.lower() for w in words if not
w.lower() in stopWords and not isInt(w) and w.isalpha() and
len(w) > 2]
        accum -= len([w for w in filtered_words if w ==
input_string])
```

```
        filtered_words_global += filtered_words
        for word in filtered_words:
            comp = levenshteinDistance(input_string,word)
            accum += float(comp) / len(words)
        return accum

    countWordsInSubChapter = memoize(countWordsInSubChapter)

    def chapterPlusIndex(chapter):
        '''
        Function used to return the (sub)chapter with the name
and text closest to the
        input by comparing their levenshtein distances with a
created index which is more
        accurate than the other chapters or SubChapters
        '''
        index = levenshteinDistance(input_string, chapter.name)
        if type(chapter) is SubChapter:
            index += countWordsInSubChapter(chapter)
        elif type(chapter) is Chapter:
            accum = 0
            for subchapter in chapter.subchapters:
                accum += countWordsInSubChapter(subchapter)
            accum = accum/len(chapter.subchapters)
            index += accum

        return (float(index), chapter)
    xs = map(chapterPlusIndex, chapters)
    return minimumChapter(xs)

def memoize(f):
    '''
    Function used to make searching the database quicker in the
"bestChapter" function by repeating
    a code that has been ran before in the same way so it takes
less time to run it again for another
    chapter
    '''
```

```
    memory = dict()
    def memoizedFunction(x):
        if x in memory:
            return memory[x]
        else:
            memory[x] = f(x)
            return memory[x]
    return memoizedFunction

def getChapter(input_string,book):
    '''
    Function used to find the chapter or subchapter that is
closet to the input by comparing the input
    to every chapter and SubChapter with the "bestChapter"
function
    '''
    return bestChapter(input_string, list(chain(*[c.subchapters
for c in book.chapters])) + book.chapters)

def levenshteinDistance(str1,str2):
    '''
    A function which takes two string and converts them into
unicode, and then compares them with the "levenshtein"
    function after making them lowercase and removing periods
from the strings
    '''
    str1, str2 = unicode(str1), unicode(str2)
    return
levenshtein(str1.lower().rstrip('.'),str2.lower().rstrip('.'))

def isInt(string):
    '''
    A function used to test if a string contains numbers and
returns a list of those strings.
    This is used to remove numbers from the comparison of the
input to the text in SubChapters
    '''
    string = [x for x in string if x in '0123456789']
```

```
    return len(string) > 0

####### Search Data End (Functions)

####### Search Data Begin

# These stop words are words that are not related to data seach,
but used in spoken and written English
stopwords = """
a about above after again agains all am an and any are aren't as
at be because been before being below between both but by can't
cannot
could couldn't did didn't do does doesn't doing don't down
during each few for from further had hadn't has hasn't have
haven't having he
he'd he'll he's her here here's hers herself him himself his how
how's i i'd i'll i'm i've if in into is isn't it it's its itself
let's me
more most mustn't my myself no nor not of off on once only or
other ought our ours ourselves out over own same shan't she
she'd she'll
she's should shouldn't so some such than that that's the their
theirs them themselves then there there's these they they'd
they'll they're
they've this those through to too under until up very was wasn't
we we'd we'll we're we've were weren't what what's when when's
where
where's which while who who's whom why why's with won't would
wouldn't you you'd you'll you're you've your yours yourself
yourselves first
second twice third fourth - + one two three four five six seven
eight nine ten john many times much will can miles years cent
men hours
exercise man must
"""
stopWords = stopwords.split()

def search(input_string,book):
```

```
    '''
    Main function which uses the functions above to compare an
input to the
    book's chapters and subchapter, and returns the chapter with
the closest
    related information
    '''
    filtered_words_global = []
    chapter = getChapter(input_string,book)
    return displayPages(chapter.pages)

def displayPages(pages):
    '''
    Function used to return the chapter's pages file names which
are
    used to find the file locations of the book's pictures
    '''
    pages = sorted(list(pages))
    return [os.path.join('Math6-%03d.png'%(x)) for x in pages]

####### Search Data End
'''
Statement used by server to run the search using an input sent
from the webpage
'''
if __name__ == '__main__':
    createNewMathOut()
    book = readBook()
    search(u'',book)
```

```
'''
Authors: Samuel J. Gervais and Thomas R. Curtin
School: Saint Pius X High School
Email: samgervais512@gmail.com
Description: Uses Tornado software to create a server for the
webpage to run on.
'''
import tornado.ioloop
import tornado.web
from mimetypes import MimeTypes
import urllib
import tornado.escape
import data
import json
import os.path

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        f = open("../HTML/index.html")
        s = f.read()
        self.write(s)

class HTMLFileHandler(tornado.web.RequestHandler):
    def get(self,fileName):
        url = urllib.pathname2url(fileName)
        t = MimeTypes().guess_type(url)
        print t
        self.set_header("Content-Type", '' + t[0] + ';
charset="utf-8"')
        f = open("../HTML/" + fileName)
        s = f.read()
        self.write(s)
class MathFiles(tornado.web.RequestHandler):
    def get(self,fileName):
```

```
        url = urllib.pathname2url(fileName)
        t = MimeTypes().guess_type(url)
        print t
        self.set_header("Content-Type", '' + t[0] + ';
charset="utf-8"')
        f = open("../IPython/tranMath6/" + fileName)
        s = f.read()
        self.write(s)

class QueryHandler(tornado.web.RequestHandler):
    def get(self, query):
        print "searching"
        data.createNewMathOut()
        book = data.readBook()
        results = data.search(query,book)
        print "here"
        self.write(json.dumps(results))

application = tornado.web.Application([
    (r"/", MainHandler),

(r"/(index\.css|index\.js|Digital%20Aristotle\.png|DA\.png|doge\
.png|favicon.ico)", HTMLFileHandler),
    (r"/(Math6-[0-9]+\.png)", MathFiles),
    (r"/search/(.*)", QueryHandler)
])


if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

```
<!--
Authors: Samuel J. Gervais
School: Saint Pius X High School
Email: samgervais512@gmail.com
Description: Program to creat webpage with attached JavaScript
and CSS files
-->
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <title>Digital Aristotle</title>
    <link rel="icon" type="image/png" href="DA.png">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.m
in.js"></script>
    <script type="application/javascript"
src="index.js"></script>
    <link rel="stylesheet" type="text/css" href="index.css">
  </head>

  <body>
  <img id="displayImg" src="Digital Aristotle.png">
  <br><br>
  <input type="text" placeholder="Search" autocomplete="off"
id="search">
  <div id="output">
  </div>
  </body>
</html>
```

```
// Authors: Samuel J. Gervais and Thomas R. Curtin
// School: Saint Pius X High School
// Email: samgervais512@gmail.com
// Discription: Program for the main functions of the search bar
which sends search to server, and displays the results.
$(document).ready(function() {
    $("#displayImg").hide().show(1000);
    $("#search").delay(1000).hide().slideDown(1000);
    $("#search").change(function() {
       var search=$("#search").val();
       search = $.trim(search);
       if(search == "doge") {
       $("#output").empty()
             $("#output").prepend('<img src="doge.png"
/>').hide(100).show(1000);
       }
       else if(search == "acknowledge") {
             var random = Math.floor(Math.random()*10);
       console.log(random)
             var acknowl = [
             "http://jordanemedlock.com",
       "http://www.cgpgrey.com"
             ];
       if(random % 2 == 0) {
         window.open([acknowl[0]]);
       }
       else {
         window.open([acknowl[1]]);
       }
       }
       else {
             $.getJSON("/search/" + encodeURIComponent(search),
function(data) {
         console.log(data);
         $(output).empty()
         for (var i = data.length - 1; i >= 0; i--) {
           console.log(data[i]);
```

```
            $("#output").prepend('<img src="' + data[i] + '"/>');
          }
        })
        }
  })
})
```

```css
/*
Authors: Samuel J. Gervais and Thomas R. Curtan
School: Saint Pius X High School
Email: samgervais512@gmail.com
Discription: The CSS program for the aesthetics of the webpage.
*/
body {
  text-align: center;
  background-color: #DDD
}

#search {
  height: 10%;
  font-size: 40pt;
  border-radius: 10px;
  outline-width: 0;
  width: 70%;
}

#displayImg {
  width: 50%;
}

#output {
  display: block;
  margin-left: 0 auto;
  margin-right: 0 auto;
  text-align: center;
  overflow-x:scroll;
  position: relative;
  width: 100%;
  height: 100%;
  font-size: 20pt;
  display: inline-block;
}
```