*Digital Aristotle: Methods, Problem Statement, Results*

**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: HTML (HyperText Markup Language), JQuery, and CSS (Cascading Style Sheet) files. The HTML file loads the scripts for JQuery, 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 JQuery 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 JQuery 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 MIME (Multi-Purpose Internet Mail Extensions), 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. The 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 uses the user’s input which is sent to the server from the JQuery 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 creating a variable that contains the Book object, and another that contains a JSON document based off of the Book which has the 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 is 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.

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. 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. **[Create an equation for this… Jordan, plz halp]** 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.

**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.

**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 operable.