NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

NAVY WATCHSTANDER TRAINING AND TESTING TOOL

by

Johnathan V. Mooring

June 2018

Thesis Advisor: Neil C. Rowe
Co-Advisor: Loren E. Peitso

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE NAVY WATCHSTANDER TRAINING AND TESTING TOOL	**5. FUNDING NUMBERS**
6. AUTHOR(S) Johnathan V. Mooring	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

This thesis investigates the automated creation of operator study and self-test materials directly from standardized operating manuals. The targeted operating manual series is the Engineering Operational Sequencing System (EOSS) instructions. The developed software scans the standard Portable Document Format (PDF) versions of the manual, analyzes the contents, autonomously generates study and test materials, and prepares software feedback to help students and watchstanders study the required material more effectively. The entire process takes place in real-time and is based on standard Navy operating-document design patterns. The employed principles and techniques should be applicable to other types of procedural documents in the Navy and throughout the Department of Defense (DoD).

14. SUBJECT TERMS EOSS, automated quiz generation, automated learning tools, flashcards, multiple choice, steps ordering, intelligent tutoring systems	**15. NUMBER OF PAGES** **105**
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

NAVY WATCHSTANDER TRAINING AND TESTING TOOL

Johnathan V. Mooring
Lieutenant Commander, United States Navy
BS, Texas A & M University, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2018

Approved by: Neil C. Rowe
Advisor

Loren E. Peitso
Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates the automated creation of operator study and self-test materials directly from standardized operating manuals. The targeted operating manual series is the Engineering Operational Sequencing System (EOSS) instructions. The developed software scans the standard Portable Document Format (PDF) versions of the manual, analyzes the contents, autonomously generates study and test materials, and prepares software feedback to help students and watchstanders study the required material more effectively. The entire process takes place in real-time and is based on standard Navy operating-document design patterns. The employed principles and techniques should be applicable to other types of procedural documents in the Navy and throughout the Department of Defense (DoD).

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

CS	Computer Science
CSOSS	Combat Systems Operational Sequencing System
DDG	Guided Missile Destroyer
DoD	Department of Defense
DTD	Document Type Definition
HTML	HyperText Markup Language
EOOW	Engineering Officer of the Watch
EOSS	Engineering Operational Sequencing System
EPCC	Electrical Plant Control Console
FOUO	For Official Use Only
GUI	Graphical User Interface
MB	Megabytes
NAVSEA	Naval Sea Systems Command
NLP	Natural Language Processing
NT	Name, Text
NNT	Notes, Name, Text
NoDoSE	Northwestern Document Structure Extractor
NSSC	Naval Sea Systems Command
NTT	Navy Training Tool
PACC	Propulsion and Auxiliary Control Console
PDF	Portable Document Format
VSQ	Visual Question Generation
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The military has procedures and documents that explain to a service member how to respond to different situations. These procedures range from step-by-step guides to general guidance. No single tool allows personnel to review their understanding prior to a formal test or an actual event. Individual military organizations do not have the time, money, or personnel to develop review materials for every manual, let alone every procedure. An automated means to take a procedure as input and present it to users as study and review materials is essential.

The required tasks for military personnel have diversity depending on the service branch, specific objective, and location. To accomplish these tasks, the military has a wide range of equipment and personnel. Those personnel require training and study to enable them to perform their different roles and operate specific equipment. To remove the uncertainty of background and familiarity with equipment and processes, the military produces documents giving standard procedures. These procedures define steps on how to accomplish a specified task with consistent positive results.

In Navy Engineering in particular, the Engineering Operational Sequencing System (EOSS) directs watchstanders' actions. This series of documents mandates the steps to perform when engineering sailors operate their shipboard equipment. The EOSS contains both routine operation and emergency procedures. Due to unique equipment configuration, an EOSS is created for each ship. The Naval Sea (NAVSEA) Systems Command is the Navy command that creates the EOSS documents.

Sailors performing a supervisory role must be ready to respond to any casualty that arises. In EOSS, Emergency Actions are by-memory directions for the steps to be executed immediately upon casualty discovery. Once those steps are complete, the watchstander will refer to the EOSS to review and verify that the correct steps were performed, as well as move onto the procedure's recovery portion. Each Navy engineering watchstander memorizes the pertinent Emergency Actions associated with their assigned watch station; the by-memory learning portion is critical to both the safe

operation of the ship and generating the appropriate roster of watch personnel to operate the ship. Verifiably accurate tools must be available to learn these procedures quickly and to verify that knowledge, as the 19th-century method of each watchstander creating their own set of flash cards is error-prone and lacking guarantees of accuracy and adequate task coverage.

This thesis addresses the taking of a procedural document as disseminated and extracting the text from it to use for learning tools. There are two basic challenges. The first is to extract the data, understand its relevance, and store it. The second is to create meaningful learning tools with that data. The Navy Training Tool (NTT) is a program written in the Python language that prompts a user to select a procedure to study and then generates multiple quizzes to test their knowledge. The quiz types are flashcards, multiple-choice, and steps ordering. These tools lend themselves to dealing with the steps as a unit as opposed to trying to understand the meaning of the words within the steps.

The next chapter, Chapter II, discusses previous work with parsing and generating questions. Chapter III presents the methodology used to address the two challenges of data extraction and learning tool construction. Chapter IV covers the results of implementing the methodology. Chapter V presents the conclusions obtained from our work and possible future work in this research area.

II. BACKGROUND

A. TEXT GENERATION

Examining grammars and parse trees is essential for understanding how text is arranged to enable extracting and generating additional data/metadata.

1. Grammars

Grammars are a way to model text that follows precise rules and behaviors, such as communication languages and computer programs (Kandar, 2013). Grammars are defined by a starting symbol, the set of grammar rules, the set of non-terminal symbols, and the set of terminal symbols (Kandar, 2013). Grammar rules allow forming new strings from any other character sequence in the language by following a set of constraints.

EOSS documents can also be defined by a grammar. Figure 1 shows some grammar rule examples that apply to EOSS.

Start -> Header, Details, Steps

Header -> Header Label, Header Text, Header | Header Label, Header Text

Details -> Details Header, Details Steps, Details |empty string

Key
A -> B: the state A is replaced by B terms
C, D: comma separated terms are present together, CD
E | F: represents a choice between E and F

Figure 1. Grammar Production Examples

EOSS documents have three components strictly ordered: header, details (which are optional), and steps. The header consists of a series of bold header labels and

associated text. The details section consists of a series of text items and associated steps. Steps can have children steps. Steps can have notes and warnings prior to a step's text.

2. Parse Trees

Parse trees are a visual representation of how a sequence of terminal symbols is formed by a grammar. The root node at the top of the tree represents the complete string. All branches apply a grammar rule to take the leaf node values and combine them into the resulting complete string located at the root node. Figure 2 is a parse tree for three consecutive EOSS steps.

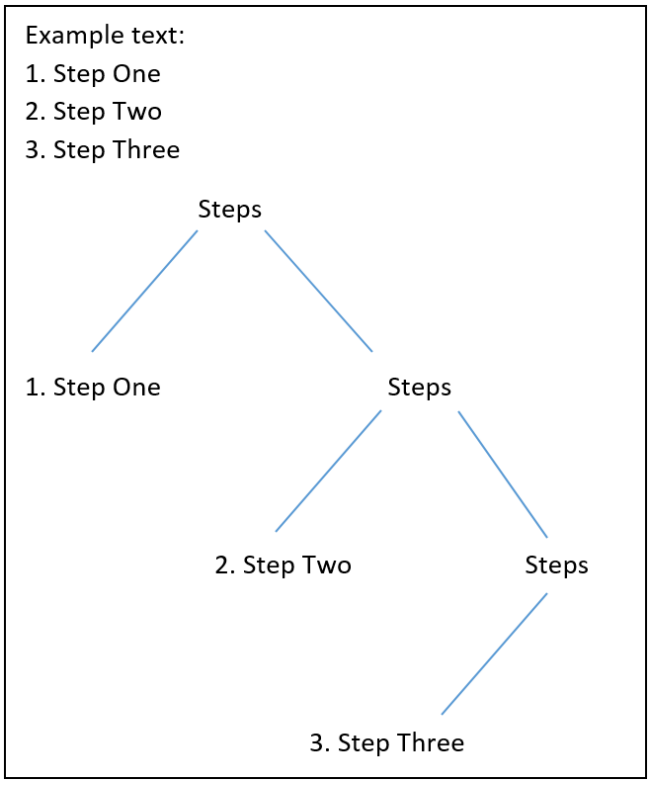

Figure 2. Sample EOSS Parse Tree for Consecutive Steps

B. TEXT PROCESSING

Before parsing, data chunks are first retrieved using specific patterns or characters to break the data into smaller parts. This is a common technique known as tokenizing.

Tokenizing does not examine the semantics of the data chunks. For EOSS parsing, the text of the steps of the EOSS procedures, which are the primary learning target, were not partitioned further.

Shallow parsing recognizes some word patterns, but does not attempt a complete parse tree (Turenne, 2013; Thanaki, 2017). Shallow parsing builds key relationships between sentence components that satisfy a set of requirements. In this type of parsing, knowledge is required about the major components of the text. EOSS documents can be parsed with shallow parsing that identifies the main structures.

Deep parsing is a complete analysis of the relationships between the strings in text to infer a more accurate guess as to their semantic relationships (Hardeniya, 2015). Deep parsing can employ semantic rules and procedures as well as grammars to capture all relationships between strings in some text. McCord and Boguraev (2012) used deep parsing with two tools, an English "slot grammar" parser and predicate-argument structure builder, to analyze sentences. Their approach used syntactic analysis to make parse trees, then ranked semantic interpretations and used the one with the highest rating. Work has been done to automate the handling and parsing of text documents with partly and fully structured data rather than prose. For instance, Adelberg (1998) explored how to parse a well-formed HyperText Markup Language (HTML) document. He proposed a three-step process: Choose a document model to use, map document sections to the model format, and specify and apply an output format. Research has also been conducted to create specialized parsers from examples of a category of structured text (Poruban et al., 2010).

C. PDF FORMAT

EOSS documents used in this thesis were in PDF format. Adobe Acrobat PDF is a container format that can contain text, data, and images. Each container has associated formatting data for size, placement, and font characteristics. When viewing a PDF through a Windows Notepad program, container details are available, but the text is encoded. A variety of open-source software can understand the structure and parse files in PDF format.

Fang et al. (2006) retrieved data from a PDF document by injecting tags around text found within the document. They took the free-form text from the PDF and used both position cues and pattern matching to determine tag placement. Other work extracted text from scientific-article PDFs (Ramakrishnan et al., 2012) by understanding the layout of the text. They detected blocks of text by spatial layout and then classified it (Ramakrishnan et al., 2012). They used an open-source tool called GPL to identify blocks of text.

EOSS PDFs have restrictions beyond a standard PDF, enabling a more detailed EOSS grammar than a PDF grammar. Each header field is a box with its text within. The header label is bold, while the text is not. The details and steps come after the header portion. The labels on the details and the first level of the steps are in all capitals. The details and steps have substeps, and each level is further indented from the parent text. Text spacing, positioning, boldface text, and capitalization can be useful for EOSS parsing.

D. GENERATING QUESTIONS

The second challenge of this thesis is extracting review questions from the text retrieved from the source documents. Bork (1980) states that an advantage with on-line quizzes is that the user is having a conversation with the developers to test their knowledge. Online quizzes also allow the user to receive immediate feedback on the correctness of their answers, as well as a summary upon quiz completion.

1. Flashcards

Flashcards are a widespread tool for learning association pairs. Flashcards in the physical sense are note cards that have questions on one side and answers on the other. Kupzyk et al. (2011) for sight-reading words and (Macquarrie et al., 2002) for English word and meanings use flashcards to learn paired sets of data. They studied the quantity and placement of new cards within a known deck of cards as well as the number of review repetitions. Bjordahl et al. (2014) used flashcards in teaching children math-division rules.

Another flashcard project automated the development of quizzes to learn Japanese (Dong & Liu, 2013). These allow the user to see the question, answer it, and then flip the card over to verify their answer was correct. In an electronic implementation, flashcards gives a user the ability to go between viewing a question and seeing an answer. Dong and Liu used flashcards to test associations between language phrases in English, Japanese characters (in Hiragana or Katakana), the word symbols in Kanji, and in the "English character pronunciation" of Japanese words, romaji. The user was provided feedback on their answer choices. Hürst et al. (2007) allow a user to create their own flashcard questions through an "iPod Quiz Generator." This tool permitted an instructor to type question and answer pairs to be stored as flashcards.

2. Multiple Choice

Multiple-choice questions are a good way to test understanding of text. Towns (2014) suggests some features that make for good multiple-choice questions: the questions should be short, to the point, and simply worded. The answers should be listed vertically, include the correct answer, and be of similar length.

Welbl (2017) generated multiple-choice questions from scientific documents. They employed a natural-language-processing tool, a noisy classifier, to identify good choices for questions. The noisy classifier could determine the keywords within a passage, or those that held the most meaning. After a question and correct answer were selected, a trained model was used to create good alternative choices similar in form to the correct answer.

Multiple-choice questions with random numbers were generated by a MATLAB program in (Azamov, 2015). This approach works when there is a single right answer to a problem. Multiple-choice quizzes concerning algorithms were automatically generated in (Bruce et al., 2009) to test students on the input, output, selections, algorithm, and parameters for specific algorithms. They also generated questions to match algorithms with their respective outputs.

Hurst et al. (2007) allowed a user to create their own multiple-choice questions through their iPod Quiz Generator. This tool permits the user to type in a question and the possible choices and identify the correct answer, which are then stored for future use.

3. Fill-in-the-Blank

Sakamoto (2017) created fill-in-the-blank questions for Wikipedia information. They determine the data layout for the text and calculate the value of words. Those calculations allow them to choose a word to omit and have the student supply.

III. METHODOLOGY

This thesis used EOSS Books 27 through 33 as our Navy procedural documents to test handling and parsing. Each ship has slightly different versions of the EOSS; the particular version we used comes from USS STETHEM, DDG 63. These manuals provide casualty-response actions for the Engineering Officer of the Watch (EOOW), Propulsion and Auxiliary Control Console (PACC) Operator, and Electrical Plant Control Console (EPCC) Operator. EOSS has a standard format that allows consistent handling of all its procedures. Our work used the procedural documents but not the index files from those EOSS Books. EOSS documents are "For Official Use Only" (FOUO) and will not be shown in this thesis. Figures will show generalized documents with the same formatting as found in EOSS but with different content.

There is a Document Type Definition (DTD) for EOSS (Naval Sea Systems Command, n.d.) that delineates and defines the structure of the EOSS sections. These tags, if present, can enable an XML tag parser to understand structure of the document to extract the data. For the source EOSS PDFs, the XML-like tags are unavailable through Adobe Acrobat.

Our test implementation was on a Windows Surface Pro that ran the Windows 10 operating system. For the programming, we used Python 3.6.0 due to ease of use, our familiarity of the language, and the availability of the os, time, and tkinter libraries. With the exception of the node and window framework, we developed all other Python code for the NTT. The parsing and window code are shown in Appendices B and C.

A. HANDLING EOSS DOCUMENTS

The EOSS source documents have a regular format (Figure 3). The documents contain a mixture of bold and non-bold text, all capitals or sentence case, and a numbering scheme. There are headers fields and text within boxes that indicate that all the text within is related. There are notes at the top and bottom of the pages.

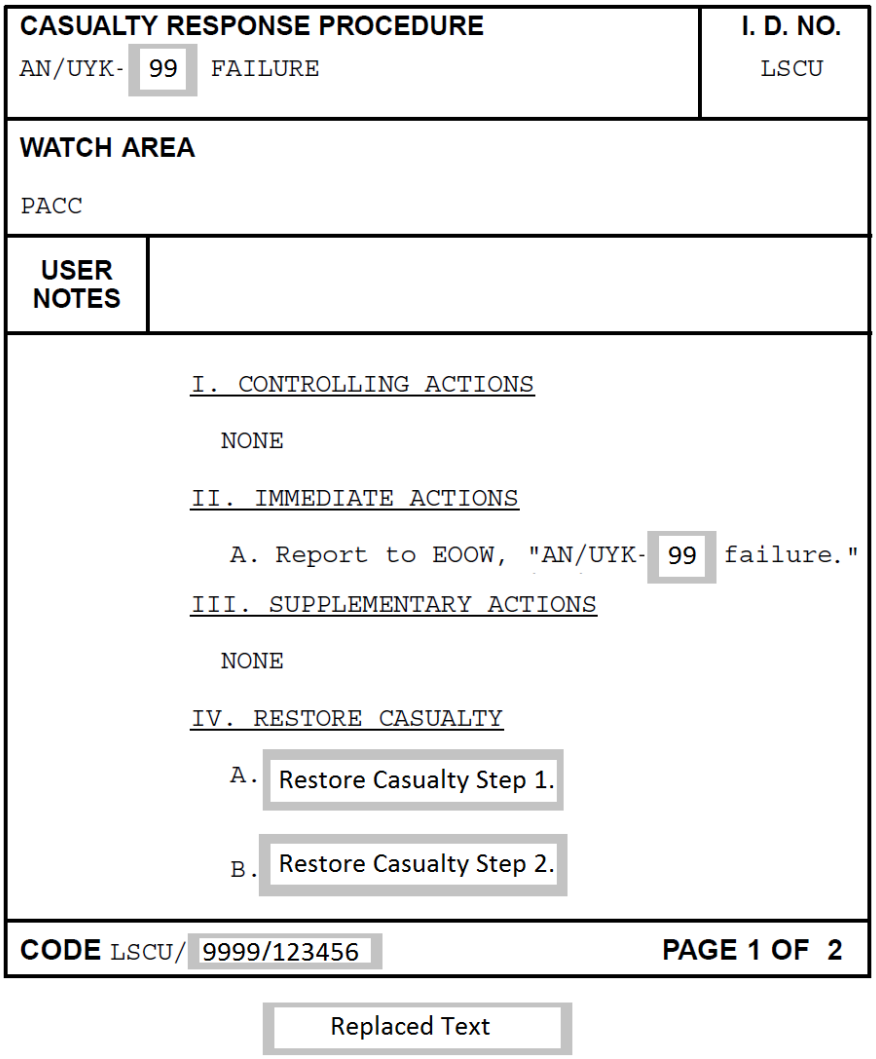

Figure 3. EOSS Excerpt

Figure 4 shows the part of the EOSS excerpt's parse tree that covers the header items. Figure 5 shows the steps portion of the parse tree, including substeps.

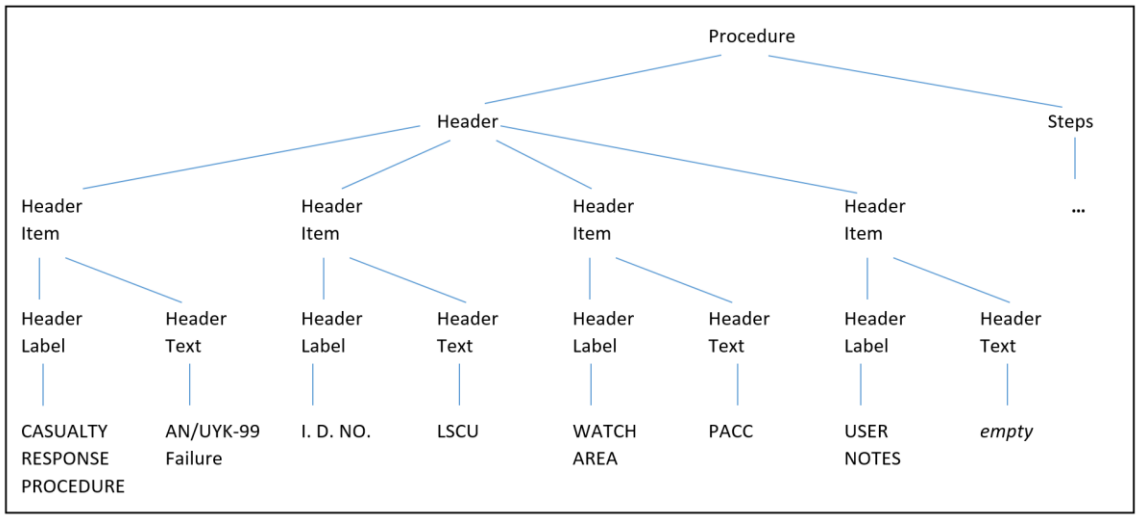

Figure 4. EOSS Excerpt Parse Tree: Header Portion

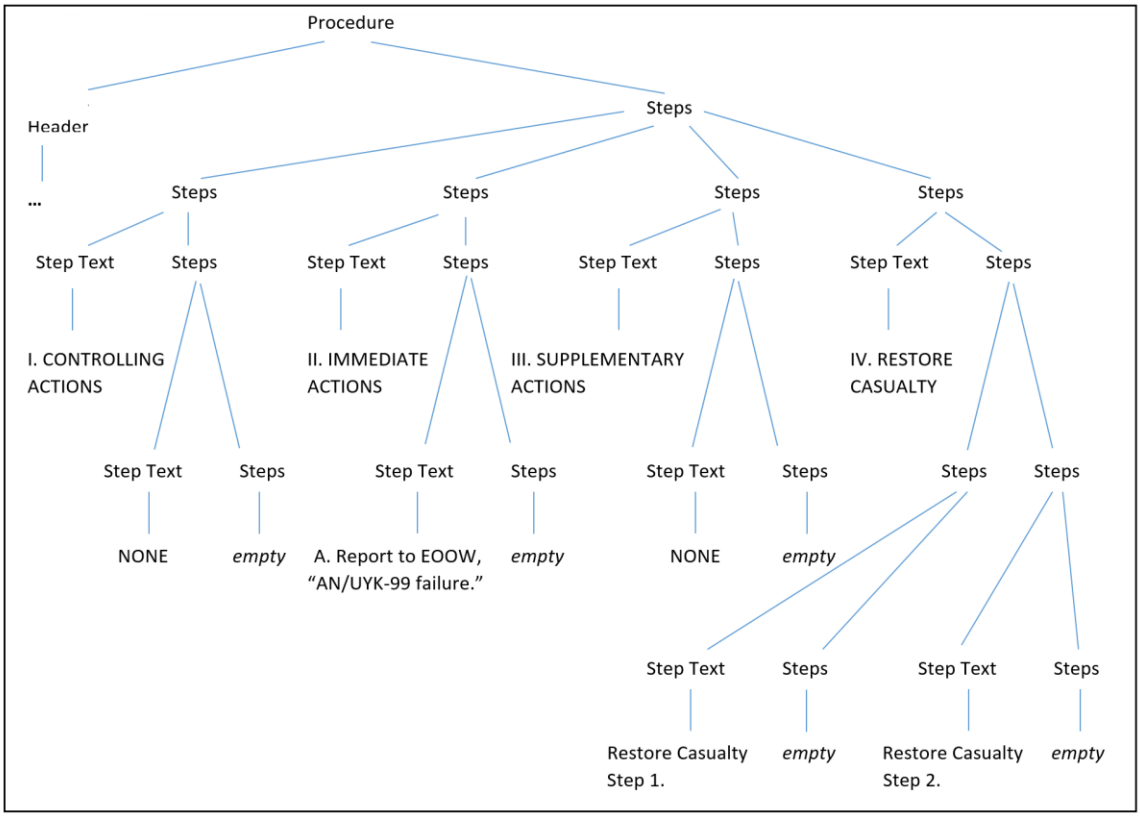

Figure 5. EOSS Excerpt Parse Tree: Steps Portion

When opening a PDF with the Windows Notepad program, PDF tags were visible, but the text was encrypted and unreadable. Similarly, Microsoft Word 2013 opens the PDF documents, but provides no way to extract text data. However, the tool mutool (Artifex Software, n.d.) available from MuPDF converts PDF format into HTML format and decrypts the text. In the resulting HTML file, useful formatting is specified such as text positioning and whether the text was bolded or not, in addition to the text itself. The **`NTTDataHandling`** function automates this process, then the **`loadTxtFile`** can change the HTML file into a text file.

After verifying the format of the HTML tags (W3Schools, n.d.), we processed the HTML data into tuples. After dividing the HTML into pages and rows, the function **`parseHTMLtextIntoTuples`** creates quadruples of (horizontal position, vertical position, bolding-status, source text). This function also trims the end-of-page comments which do not provide value for the quizzes. Due to the data conversions, special characters are temporarily rendered into their Unicode representations, and the **`restoreUnicode`** function restores those symbols to ASCII characters prior to putting data into quadruples.

B. PARSING EOSS DOCUMENTS

EOSS documents have three distinct sections. At the top is the procedure's header, as opposed to the PDF form header, which contains the information on the procedure name, procedure ID, the applicable watch station, and any User Notes. The next section gives optional details including Symptoms/Indications, Possible Causes, and Possible Effects. The last section gives the procedure's steps to execute the desired task. The text characteristics and usage distinguish the sections and allow building the EOSS grammar (Figure 6). The header is comprised of bolded and all-capital text header fields followed by not bolded text. The details portion is comprised of all-capital text followed by steps. The steps section is comprised of nested steps with step numbers. **`NTTDataHandling's generateDataStructure`** function parses the data and extracts a data structure.

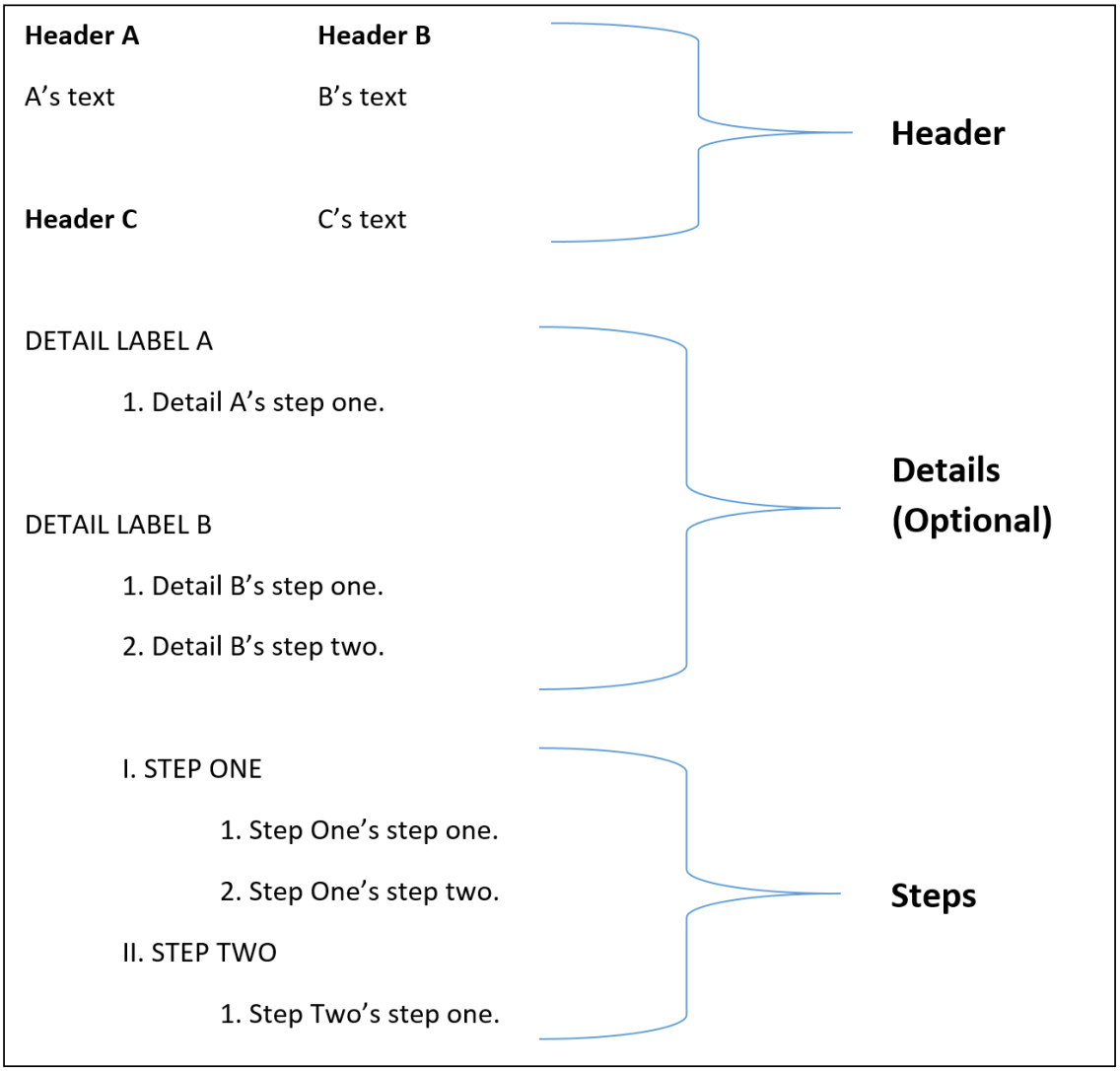

Figure 6. EOSS Sections: Header, Details, and Steps

1. Parsing the Header Portion

Each header field has a bold label followed by text that is not bolded. Either columns of empty space or blank lines separate the header fields. To reason about the header fields, we used the horizontal and vertical spacing of the text, as well as whether the text was bolded. There are two patterns of the header. One is column header-field labels at the top of a column with their associated text falling under their respective label, as in Figure 7). The other pattern is a row header field where the header field label is

followed by its text, as in Figure 8. The text pattern is determined by comparing the previous vertical value lastHeight to the vertical value of the current tuple currentHeight.

Column Header A	**Column Header B**
A's text	B's text
A's text continued	B's text continued

Figure 7. Column Header Field Example

Row Header A	A's text
	A's text continued
Row Header B	B's text
	B's text continued

Figure 8. Row Header Field Example

Header information at the top of every page repeats the procedure's name. This simply confirms we are within the same procedure. The header data structure consists of a single layer of information, header-field labels, and their associated text. We exit the header processing if we detect a note, step, or detail. If it is a step, then it has a step number and the text is not bolded. If it is a note, then the text is not bolded, led by all-capitals text, and is followed by ":" and more text. If it is a detail, the text is in all capitals and is not bolded and the next tuple has a step number.

2. Parsing the Details Portion

The details portion contains detail tags followed by a list of items. The detail tags have text that is in all capitals but is not bolded. The listed items are not bolded and in

sentence case text, and have a step number, as in Figure 9. A details step can be differentiated from a procedure step by whether it is in all capitals. (Figure 10).

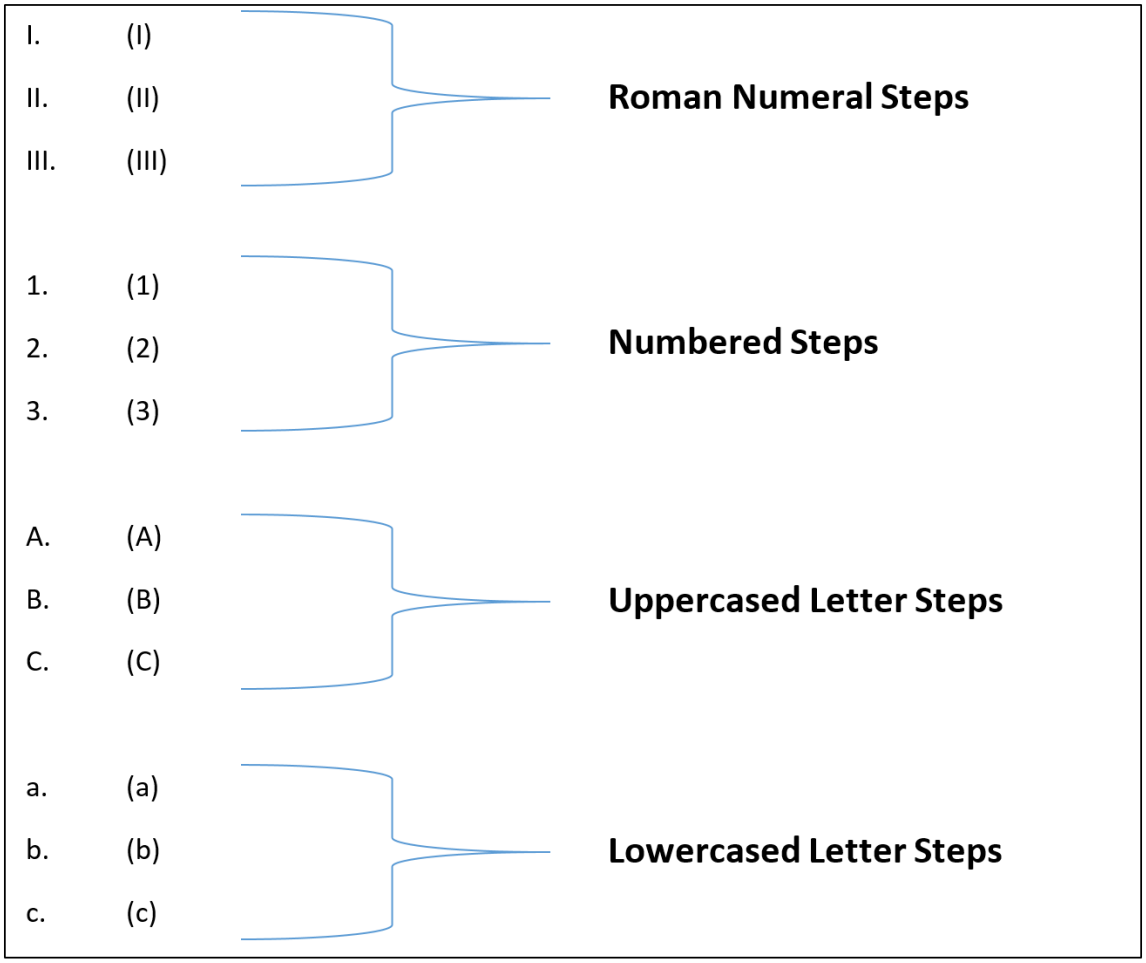

Figure 9. Step Numbers Examples

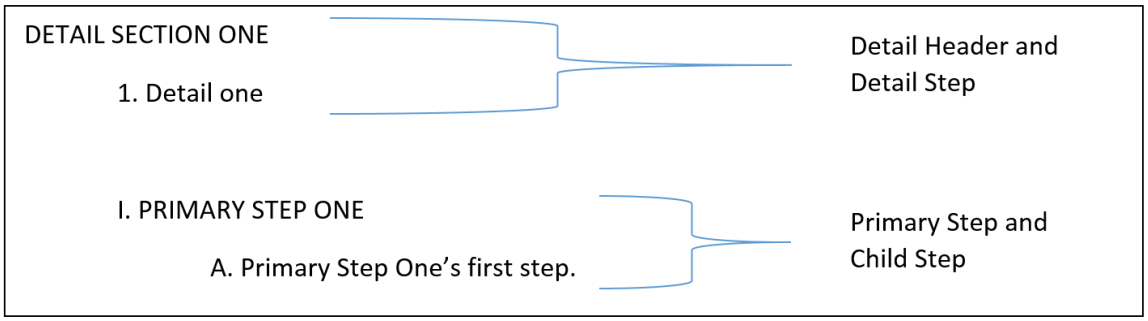

Figure 10. Detail and Step Example

3. Parsing the Steps Portion

The steps portion consists of notes and substeps. The primary substep is the highest level of the steps and is always in all-capital text. The other substeps are in sentence case. For understanding the relationships between the steps, we compared formatting and positioning. Since EOSS pages are smaller than standard sheets of paper, EOSS pages can have inconsistent positioning in the PDF documents, so positioning cannot be the only way to identify steps. To enable future handling of documents with non-EOSS numbering scheme, we did not verify that a particular step number belonged to specific type number or letter series.

We implemented a tree structure to store the information of the steps. It supports operations such as examining and navigating to parent nodes, children, assessing siblings, and printing the tree. Our **`StepsNode`** program is inspired by an example by Kromkamp (2018). We use three separate arrays to store the tree structure. The first two are **`stepTracker`** and **`stepSpacing`**, which contains information from the highest level to the lowest used. The **`stepTracker`** contains the last number of step for the respective tier, while the **`stepSpacing`** has the left value for that respective tier. The lower the tier of step, the higher the left value will be, indicating that the text is further to the right, as in Figure 11. Since all steps have positive values for indentation, we use a negative value to indicate a note in the **`stepSpacing`** array.

Tier 1:	I.		Left value = 50
Tier 2:	1.		Left value = 100
	2.		Left value = 100
Tier 3:		A.	Left value = 150

Figure 11. Step Tier and Left Value Relationship Example

To determine the relationship of a new node to the previous nodes, we compared the new left value to the last left value in stepSpacing. If the new left value is less than the last node's left value, then the step is a higher-level node. If the left values are equal, then the new node is at the same level as the last node. Otherwise, the new node is a child of the previous node. In EOSS when a parent has no children steps, "none" is used as the child step. This None step is indented, but does not have a step number.

A Note item is determined solely on formatting and content: Text is not bolded, is led by all-capitals text, and is followed by ":" and more text. Notes do not employ left values to determine their position in the tree, as they are assigned to the first step following the note.

If the current tuple was not a new step or note, then we know that it is the continuation of a previous line of text, and we concatenate the current text to the last item's text. After determining the relationship of the new node, we update the parent and child relationships.

The **`generateDataStructure`** function in **`NTTDataHandling`** returns a **`structuredArray`** representing the parsing of the given input file. It consists of header, details, and step data.

4. Running Time

The NTT's parsing code conducts one-time passes through the data. The HTML data is first broken into individual pages and rows, then quadruples are created, and then

a data structure is built from each tuple. Each piece of information is only used once per task, so the computation time is linear in the amount of data.

C. GRAPHICAL USER INTERFACE (GUI)

Tor develop a user interface for our NTT program, we used the Tkinter library (Tutorials Point, 2018). This library provides window boxes with labels and buttons, as well as pop-up text boxes. We employed Tutorial Point's Tkinter website to determine function usage and options (Tutorial Point, 2018). We used the minimal GUI application provided by Shipman (2013) to serve as the base for all of windows. To identify the tkinter elements within the text, we used the convention of bolding their names and placed angular brackets around them.

The **`NTTMain`** Python file starts the NTT program. It creates a new tkinter object called "root." Then it calls **`NTT_GUI's MainMenu`** function to generate the starting main menu window, as in Figure 12.

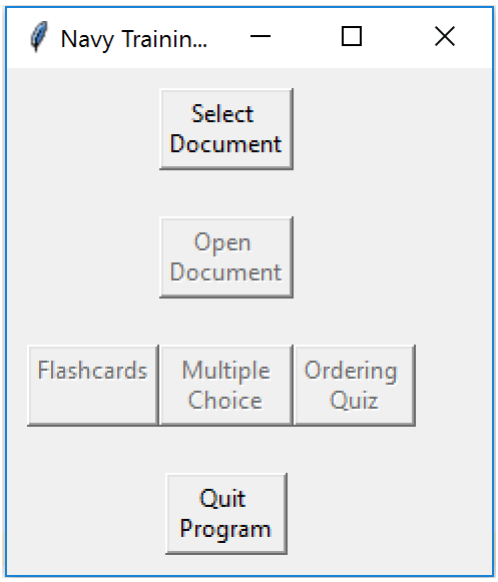

Figure 12. Initial Main Menu Window

The user clicks the <**Select Document**> button to select the procedure that they wish to practice. **`NTTDataHandling's changePDFintoDataStructure`**

function extracts the file's data. Figure 13 shows the buttons enabled after extracting the data.

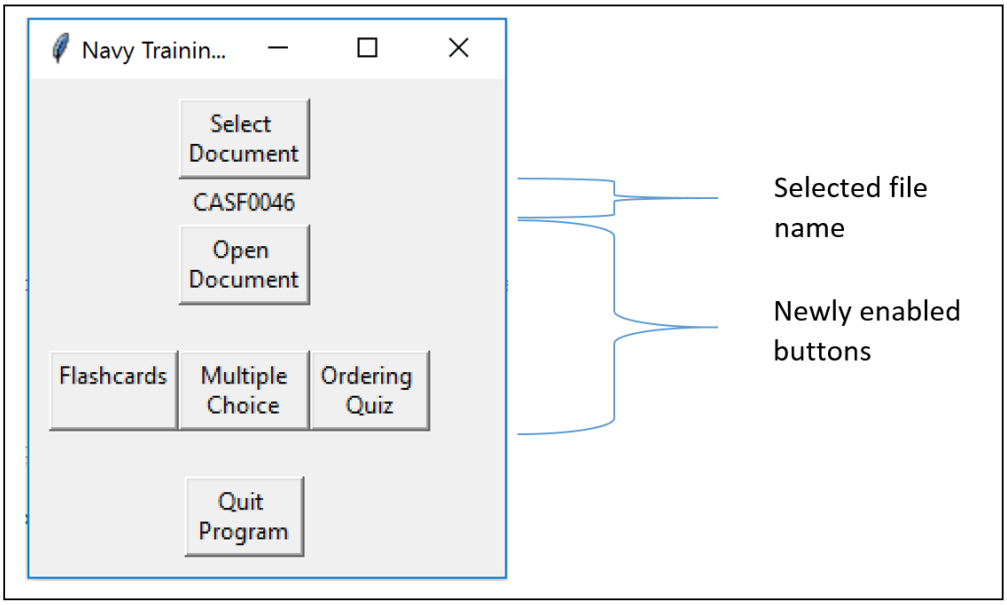

Figure 13. Main Menu Window After File Selection

D. GENERATING THE USER TESTS

Three buttons on the Main Menu window start the learning tools: **Flashcards**, **Multiple Choice**, and **Ordering Quiz**. In each learning tool, we employ left and right movement buttons, indicated by "<-" and "->," to navigate through the questions. A <**Check**> or <**Answer**> button allows the user to verify the answer to a given question. The <**Quit**> button closes the current window and returns to the main menu. We employ labels to show the questions, answers, and the completion progress. Upon reviewing and answering all the questions, we show the results displaying the time spent per question as well as their accuracy.

<**Flashcards**> chooses a step and ask what comes after it to create question-and-answer pairs. These pairs are built by traversing the steps tree and pairing steps with the next step. The flashcards are for review, and the user can compare their answer to the correct one.

Users can also answer multiple-choice questions to identify the step that follows the question step. The provided answers are the correct next step and randomly selected incorrect choices. Every time the window starts, the question answers are selected and shuffled. When the <**Check**> button is clicked, the user's choice is compared to the correct answer. The correct answer is marked in green and incorrect choices are marked in red. After answering all questions, the user can view results.

<**Ordering Quiz**> gives the user multiple steps that they must place in the correct order. The step choices, and later the answers, are positioned on the left, and the user's choices are radio buttons on the right.

Each step has associated notes. These notes apply to the given step and its children. When constructing the questions, these notes are also extracted along with the answers. None of the learning tools employs the notes, but they are available for future development.

Each learning tool employs the steps tree to construct their questions. This process requires a full transversal of the tree structure. According to Cormen et al. (2009), visiting all nodes of a tree takes $n \log n$ time. The n represents the number of nodes, while the $\log n$ portion is the position of a given node within the tree. This timing held true the test EOSS documents.

IV. RESULTS

The Navy Training Tool (NTT) we have written employs the strategies and techniques described in the last chapter. This program handles the source PDF document procedure parsing and the learning tools. Upon selecting an EOSS document, the Main Menu window allows opening the document and using on the learning tools, as in Figure 14. Figure 15 and Figure 16 show an example, which is found in Appendix C and is derived from Country Living (2017). Figure 17 shows what the program extracted and stored.

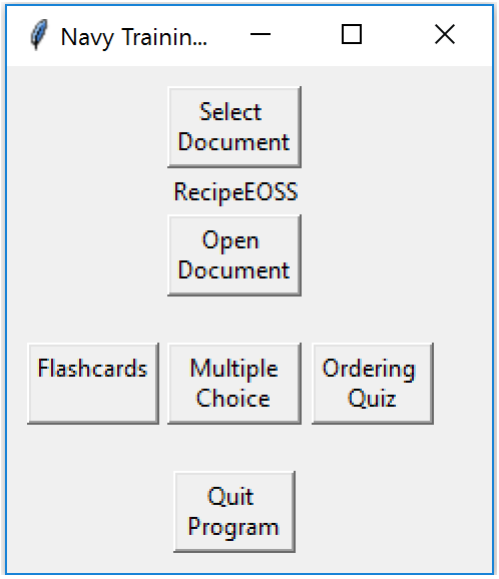

Figure 14. NTT Main Menu After Selecting Document

RECIPE PROCEDURE **I. D. NO.**

VANILLLA CAKE VCAKE

JOB TYPE

COOK

DISTRIBUTION STATEMENT: Family's secret recipe, do not distribute.

USER
NOTES

INGREDIENTS

1. Flour
2. Baking powder
3. Salt
4. Butter
5. Sugar
6. Eggs
7. Vanilla extract
8. Milk

TOOLS

1. Cake pan
2. Mixer
3. Cooler rack
4. Oven

I. PREPARATION

A. Heat oven to 400 degrees F.

B. Coat pan with butter and dust with flour.

Figure 15. Recipe EOSS Page 1

USER NOTES VANILLA CAKE

C. Sift flour, baking powder, and salt into bowl.

II. MIXING INGREDIENTS

NOTE: MIXING DONE BY MIXER.

A. Use mixer to mix ingredients.

1. Add butter.
2. Add sugar.
3. To add eggs:
    a. Crack egg
    b. Pour egg in
    c. Throw away shell.
4. Add vanilla.
5. Add milk.

B. After thoroughly mixed, pour into cake pan.

WARNING: UNDERCOOKED FOOD MAY RESULT IN ILLNESS.

III. BAKE

A. Place cake pan in oven for 30 minutes.

B. Remove cake to cool upon completion of cooking time.

IV. ENJOY

A. When cake is cool cut a piece of cake.

B. Eat your cake.

CODE VCAKE/0123/123456 **PAGE 2 OF 2**

Figure 16. Recipe EOSS Page 2

```
The following is the Header data:
['RECIPE PROCEDURE', 'VANILLLA CAKE ']
['I. D. NO.', 'VCAKE ']
['JOB TYPE', 'COOK ']
['DISTRIBUTION STATEMENT:', "Family's secret recipe, do not distribute. "]

The following is the Detail data:
['INGREDIENTS', ['1. Flour', '2. Baking powder', '3. Salt', '4. Butter', '5. Sugar', '6. Eggs', '7. Vanilla extract', '8. Milk']]
['TOOLS', ['1. Cake pan', '2. Mixer', '3. Cooler rack', '4. Oven']]

ROOT
    1 I. PREPARATION
        1.1 A. Heat oven to 400 degrees F.
        1.2 B. Coat pan with butter and dust with flour.
        1.3 C. Sift flour, baking powder, and salt into bowl.
    2 II. MIXING INGREDIENTS
        NOTE: MIXING DONE BY MIXER.
        2.1 A. Use mixer to mix ingredients.
            2.1.1 1.  Add butter.
            2.1.2 2. Add sugar.
            2.1.3 3. To add eggs:
                2.1.3.1 a. Crack egg
                2.1.3.2 b. Pour egg in
                2.1.3.3 c. Throw away shell.
            2.1.4 4. Add vanilla.
            2.1.5 5. Add milk.
        2.2 B. After thoroughly mixed, pour into cake pan.
    WARNING: UNDERCOOKED FOOD MAY RESULT IN ILLNESS.
    3 III. BAKE
        3.1 A. Place cake pan in oven for 30 minutes.
        3.2 B. Remove cake to cool upon completion of cooking time.
    4 IV. ENJOY
        4.1 A. When cake is cool cut a piece of cake.
        4.2 B. Eat your cake.
```

Figure 17. Recipe EOSS Extracted Data

A. FLASHCARDS QUIZ

The Flashcards window for this example gives the first question, which is a primary step. Figure 18 displays the window with an <**Answer**> button, and <**Quit**> button. Clicking the <**Answer**> button shows the answer to the question Figure 19.

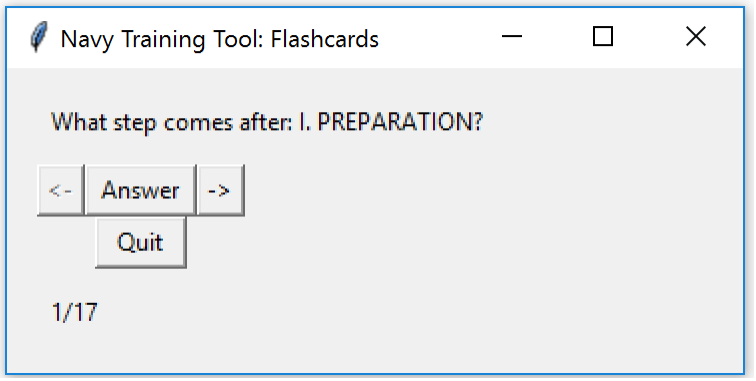

Figure 18. Starting Flashcards

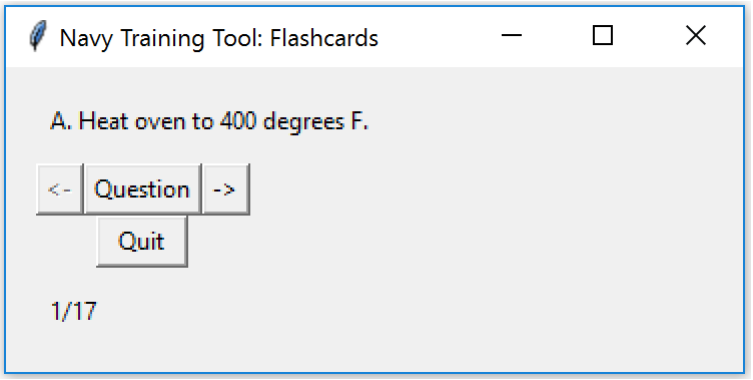

Figure 19. Flashcards: First Question's Answer

The second question (Figure 20) builds on the answer from the first question. These flashcards link the steps together to cover all the current primary step's children. If there are still more steps to be performed, a new primary step will start another set of questions. Figure 21 shows the second primary step in the Recipe EOSS.

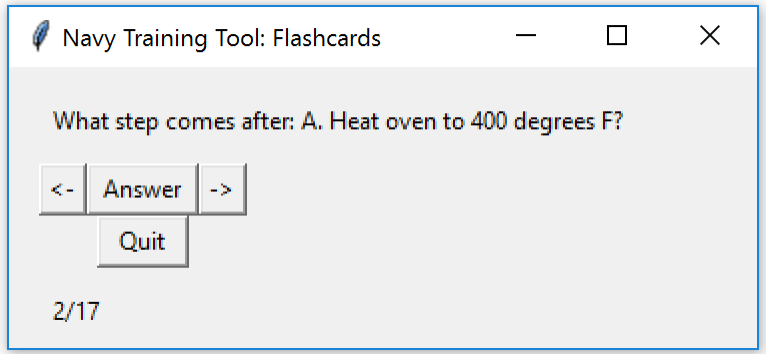

Figure 20. Flashcards: Second Question

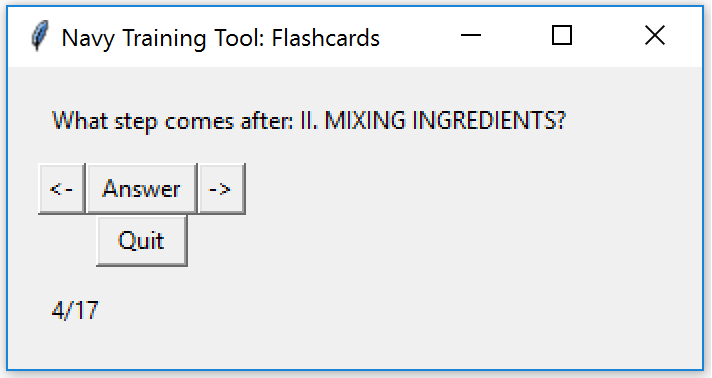

Figure 21. Flashcards: Second Primary Step

At the last question, the <**Right**> movement button's text turns into "Results" as in Figure 22. Clicking it (Figure 23) reports the time spent reviewing each question.

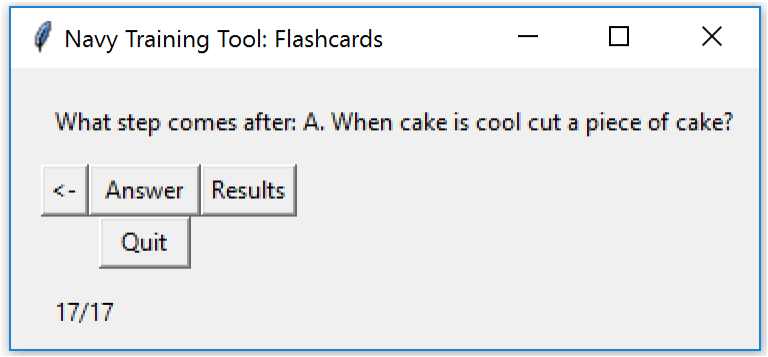

Figure 22. Flashcards: Results Button

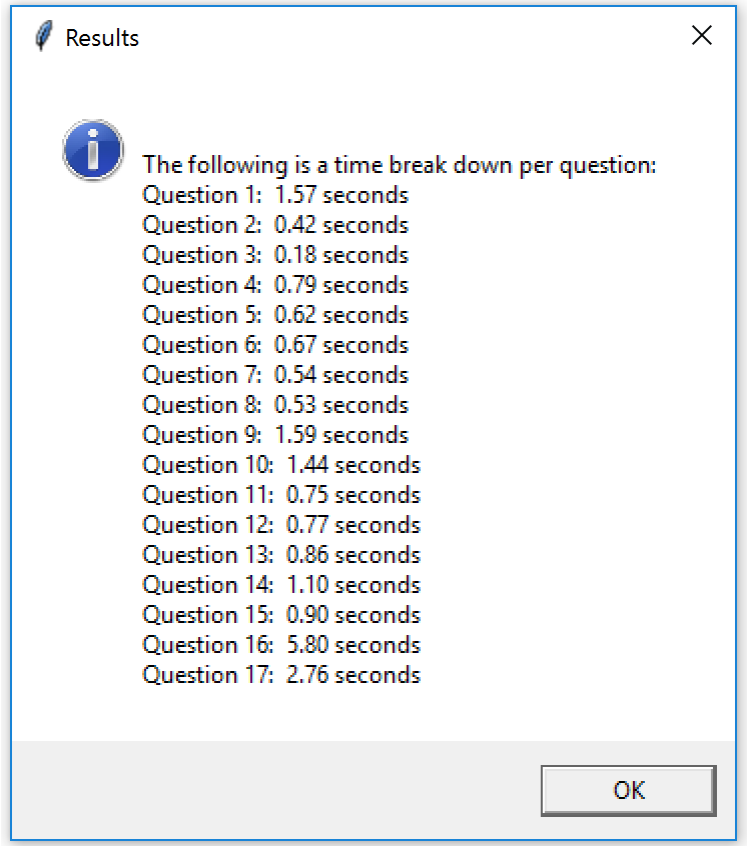

Figure 23. Flashcards: Results

B. MULTIPLE-CHOICE QUIZ

From the Main Menu window, clicking the <**Multiple Choice**> button generates a windows like Figure 24. Clicking the <**Check**> button when the correct answer is selected will turn the step green, illustrated in Figure 25. Figure 26 shows an incorrect answer choice in red. NTT stores whether a question has been answered correctly to allow the correct coloring of choices when navigating to previously answered questions. When all questions have been answered, the <**Check**> button's text changes to "Results" as with Flashcards.

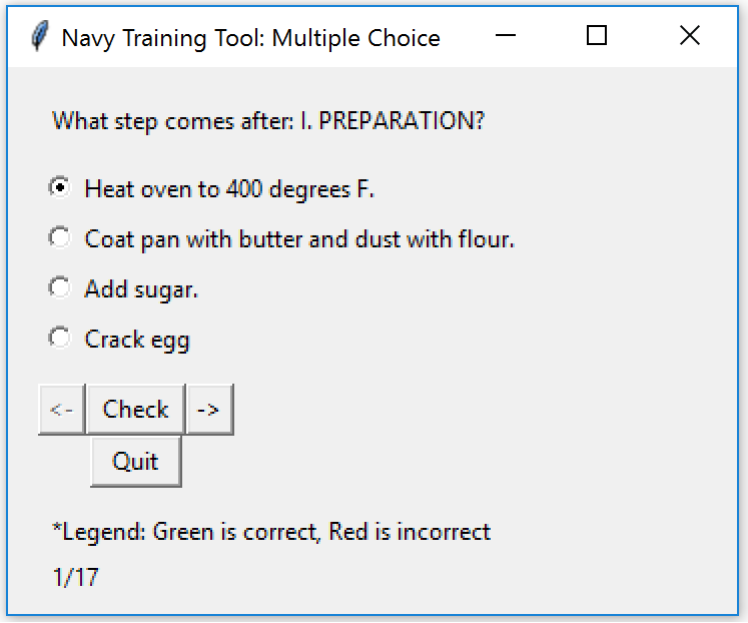

Figure 24. Multiple Choice: First Question

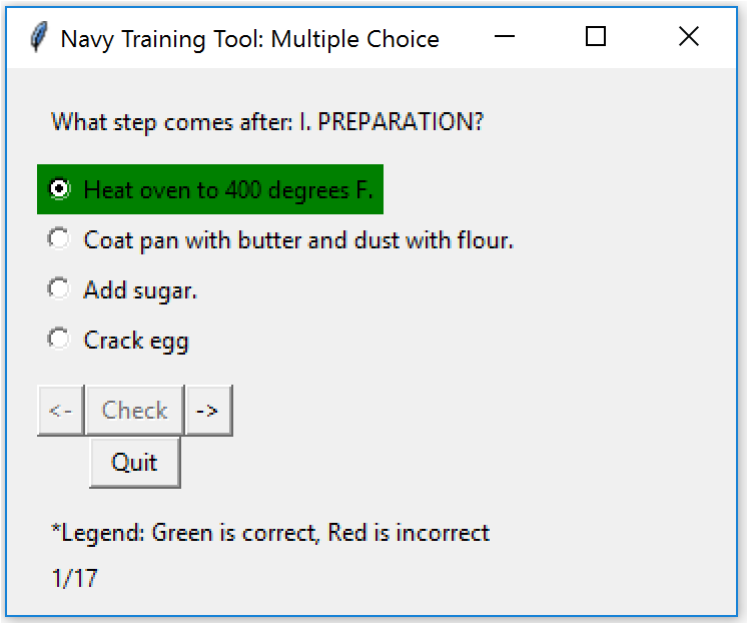

Figure 25. Multiple Choice: Correct Answer

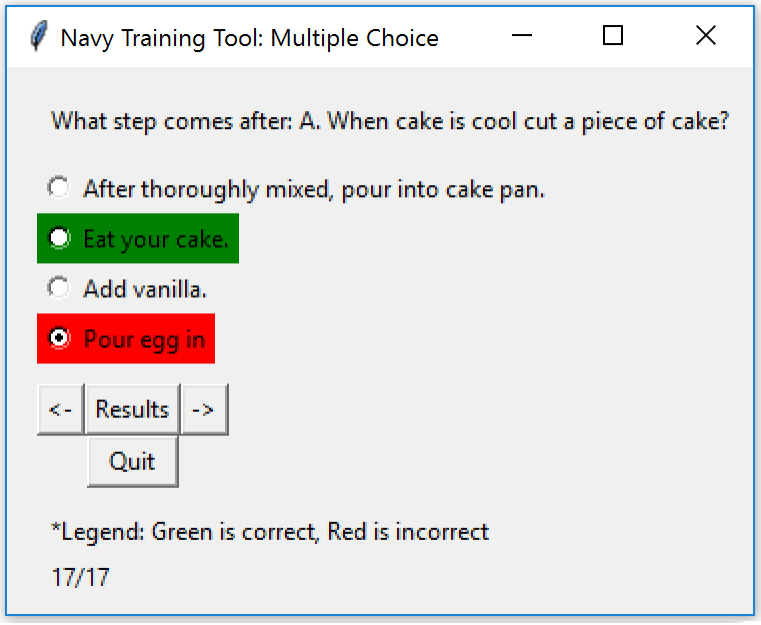

Figure 26. Multiple Choice: Incorrect Answer

When all questions have been answered, the <**Check**> button's text changes to "Results." Clicking the <**Results**> button brings up the results pop up box. These results show the time spent on each question as well as the overall number of correct answers.

C. ORDERING QUIZ

The <**Ordering Quiz**> window is shown in Figure 27. Each question has between three to six choices, as determined by the **`MAX_STEPS`** and **`MIN_STEPS`** in **`NTTConstants`** file, which are displayed by the <**Step Options**> buttons. All steps provided are valid and must be selected through clicking on the left <**Step Options**> buttons. When chosen, the step will populate on the right of the window (Figure 28) and then disable the clicked <**Step Options**> button.

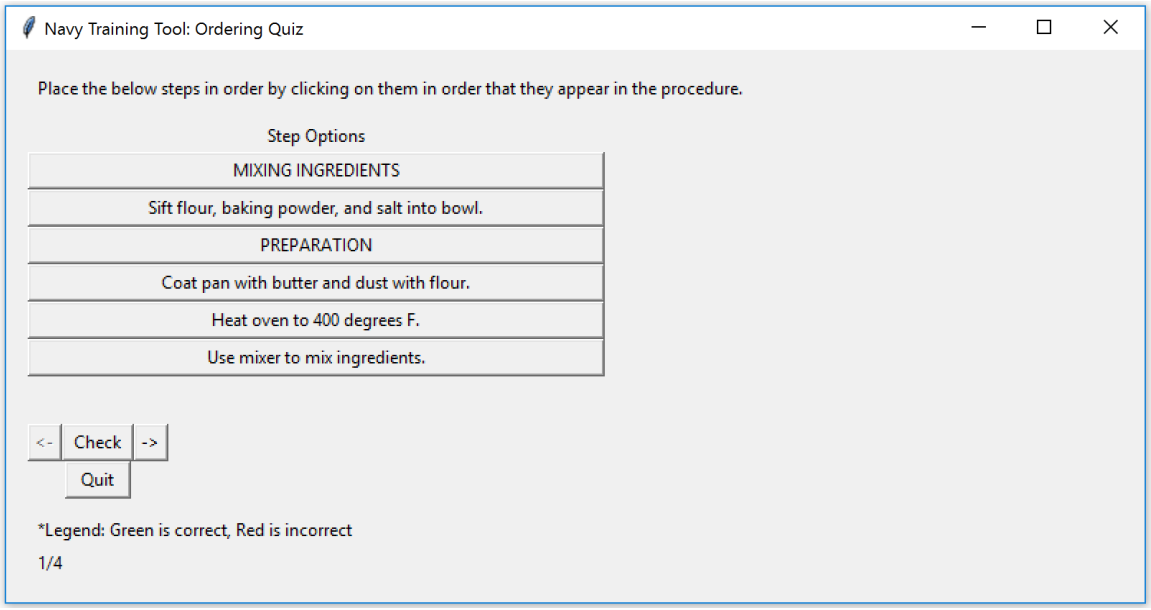

Figure 27. Ordering: First Question

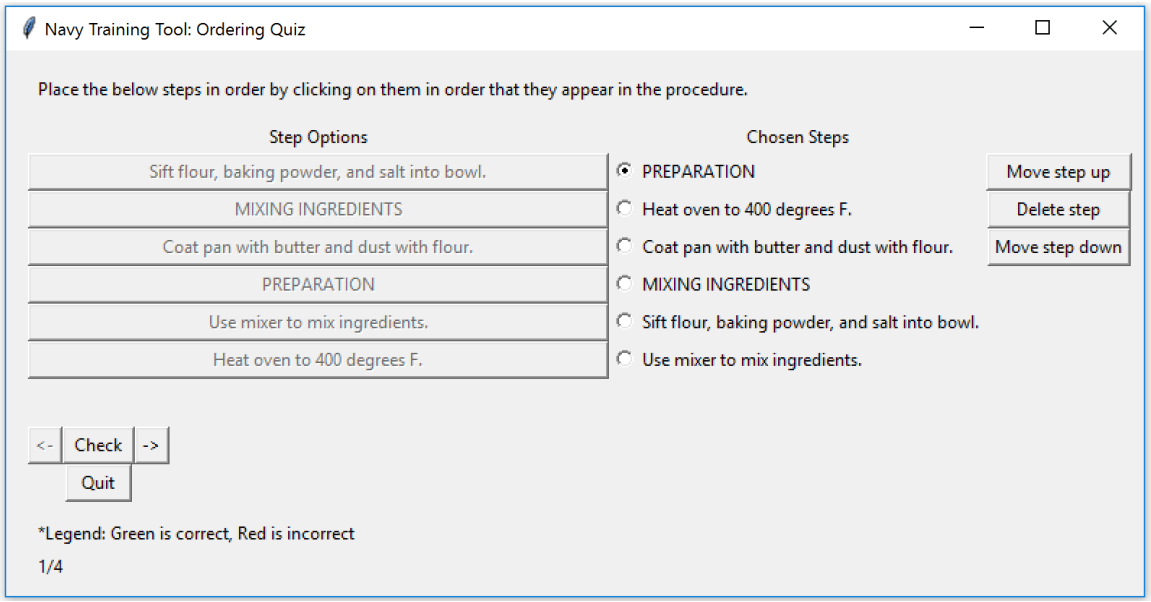

Figure 28. Ordering: After Step Selection

The movement buttons <**Move step up**> and <**Move step down**> to the right of the chosen steps, modify the placement of radio button selection of the <**Chosen Steps**>. The movement buttons can move a step up or down in the list of chosen steps. The <**Delete**> step button removes the chosen step completely.

If <**Check**> is clicked prior to all Step Options being used, then a text popup tells the user to use all the steps. Otherwise, the <**Check**> button will grade the <**Chosen Steps**> ordering, marking the correct positioning in green and the incorrect ones in red, as in Figure 29. In addition, this figure shows <**Correct Step Order**> rather than <**Step Options**>, which enables the user to compare their selection against the proper ordering.

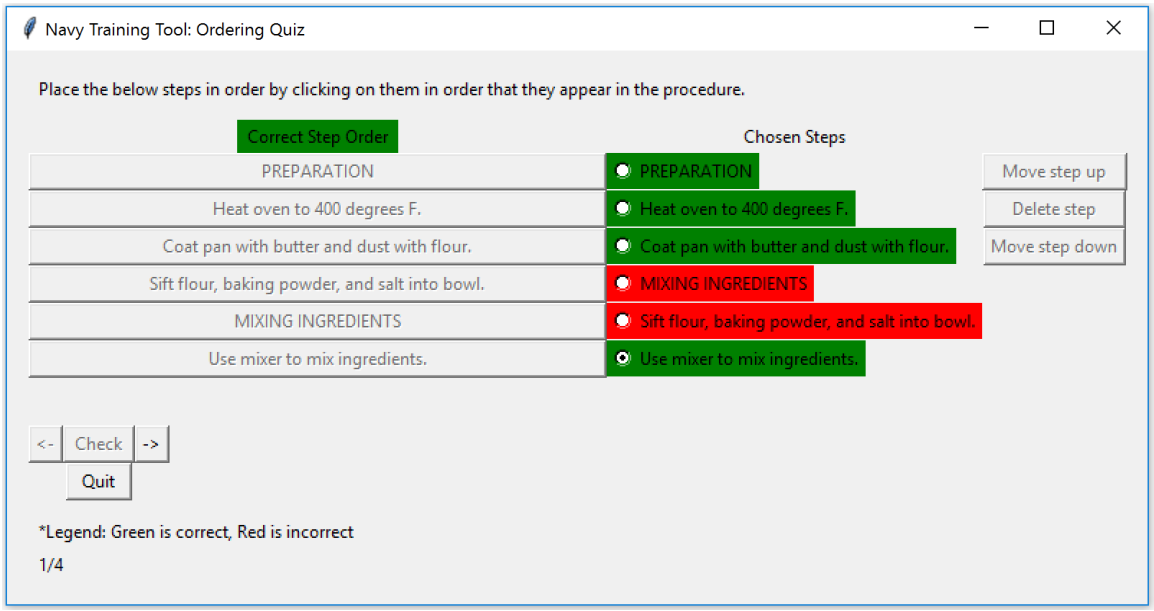

Figure 29. Ordering: Graded First Question

After all questions have been answered the <**Check**> button's text changes to "Return," as in Figure 30, and the user can click the <**Results**> button (Figure 31).

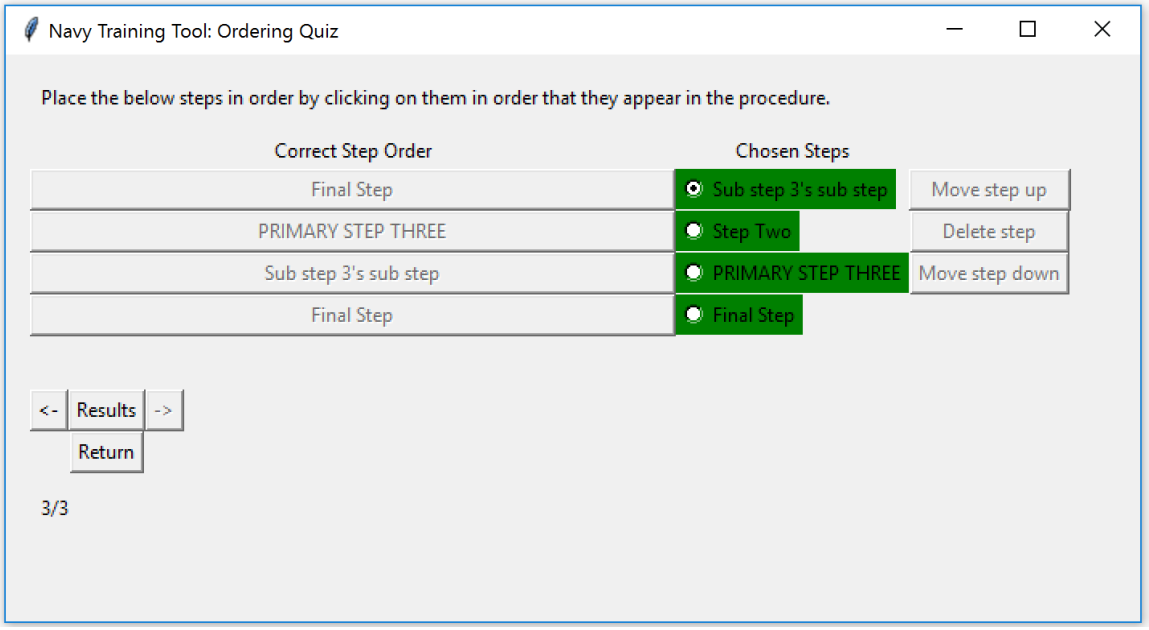

Figure 30. Ordering: Results Button

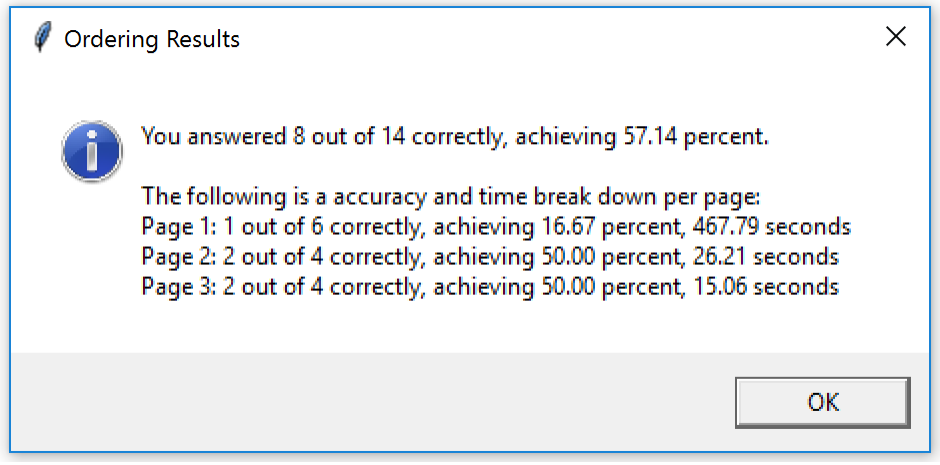

Figure 31. Ordering: Results

D. TESTING

About a dozen peers informally verified NTT's functionality and offered broad improvement suggestions. No personal data was recorded as these ad hoc tests were solely focused on capturing broad and obvious mistakes in the NTT. Discovered errors were fixed.

NTT incorporates the feedback provided. All windows have a button labeled "Quit" now, instead of "Return," to make the button's purpose more intuitive. On the ordering window after the user validates their answers, the <**Correct Step Order**> label now turns green to focus the user's attention on the newly placed correctly ordered steps. The total size of the NTT is 35 megabytes (MB). The mutool executable takes up 34 MB, while the NTT's Python source code takes up less than one MB. All NTT operations including running the program, opening the PDF, and generating the learning tools took less than a second to perform. The slowest operation was opening the PDF as NTT directs Windows to use the default PDF viewer, which requires time to open and load the document.

E. NTT PROCEDURE SUPPORT

The NTT supports the EOSS formatted documents, and was specifically developed for Books 27 through 33 covering roughly 150 procedures. Of these procedures we tested 50, ensuring to cover the full span of formatting differences. These procedures cover three supervisory watchstation's emergency actions during equipment casualties. There are differences in the number of header fields and some text casing differences within the step, while overall they have the same document structures, spacing constraints, step numberings, and text cases for identifying the detail labels and primary steps. The procedures' consistencies in horizontal spacing and grammar rules are important in enabling NTT to determine step relationships. The relationships result in a tree. A full tree transversal is a common and well understood operation, which allows the recreation of the step ordering derived from the EOSS documents. From the list of steps, then the steps are paired to obtain a valid question and answer.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

Personnel learning to perform supervisory engineering watch positions must study and memorize the emergency actions. Skills of replacement personnel must be continually retested to ensure that there are a least three people that can perform a role at any given time. This thesis built a tool to extract data from a procedural document and create three kinds of learning tools from that data: Flashcards, Multiple Choice, and Ordering. These learning tools use a tree data structure parsed from the procedural document.

The NTT will be a great asset to engineering supervisors since the EOSS is the standard document used by ships to perform engineering casualty control. An important issue with trying to memorize the EOSS is that there are no interactive tools. With the NTT and access to the EOSS digitally, a user can employ any of the three learning tools to increase their familiarity with the procedures. The ordering learning tool is the easiest with many steps listed and allows the user to organize them based on the cues found within the steps. The multiple-choice tool requires the user to be able to identify the correct answer within the choices. The flashcards tool is the hardest and requires the user to have the procedures committed to memory.

Future efforts can use more procedural documents as input including the Combat Systems Operational Sequencing System (CSOSS) and the Naval Air Training and Operating Procedures Standardization (NATOPS). Natural-language processing (NLP) could be used to determine the meaning of sentences and the words within them to provide more detailed questions. For example, the sentence “Report to the EOOW and turn the valve” could be broken down into two steps by understanding the imperative verb use as well as the employment of “and.” In addition, natural-language processing could examine multiple steps to determine the verbs and nouns for a fill-in-the-blank type learning tool.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. RECIPE EOSS DOCUMENT

RECIPE PROCEDURE **I. D. NO.**

VANILLLA CAKE VCAKE

JOB TYPE

COOK

DISTRIBUTION STATEMENT: Family's secret recipe, do not distribute.

USER
NOTES

INGREDIENTS

1. Flour
2. Baking powder
3. Salt
4. Butter
5. Sugar
6. Eggs
7. Vanilla extract
8. Milk

TOOLS

1. Cake pan
2. Mixer
3. Cooler rack
4. Oven

I. PREPARATION

A. Heat oven to 400 degrees F.

B. Coat pan with butter and dust with flour.

C. Sift flour, baking powder, and salt into bowl.

II. MIXING INGREDIENTS

NOTE: MIXING DONE BY MIXER.

A. Use mixer to mix ingredients.

1. Add butter.
2. Add sugar.
3. To add eggs:
    a. Crack egg
    b. Pour egg in
    c. Throw away shell.
4. Add vanilla.
5. Add milk.

B. After thoroughly mixed, pour into cake pan.

WARNING: UNDERCOOKED FOOD MAY RESULT IN ILLNESS.

III. BAKE

A. Place cake pan in oven for 30 minutes.
B. Remove cake to cool upon completion of cooking time.

IV. ENJOY

A. When cake is cool cut a piece of cake.
B. Eat your cake.

APPENDIX B. GUI CODE

Code available online at https://gitlab.nps.edu/jvmoorin/NavyTrainingTool/tree/master

A. NTT CONSTANTS

```
#File Extensions
PDF_EXTENSION = ."pdf"
HTML_EXTENSION = ."html"
TXT_EXTENSION = ."txt"

#HTML Tuple Index
HTML_TOP = 0
HTML_LEFT = 1
HTML_BOLD = 2
HTML_TEXT = 3

#Final Array Indices
HEADER = 0
DETAILS = 1
STEPS = 2

#Header Indices
HEADER_FIELD_NAME = 0
HEADER_CONTENT = 1

#Tree Positions
ROOT = 0

#Tree Printing Styles
NNT = "NNT" #This will have Notes, Name, then Text
NT = "NT"   #This will have Name and Text
#None will be the entire Steps Node object

#Node Note Printing Styles
TABS = "TABS"
SPACES = "SPACES"
NEW_LINES = "NEW_LINES"
SEQUENCE = "SEQUENCE"

#Roman Numerals Array (I, V, and X)
#Deal with only the upper case form
ROMAN_NUMERALS = ["I," "V," "X"]

#Not roman numerals
#"C," "D," "L," and "M" are Roman numerals, but are higher than
#  what we intend to use.
#Deal with only the upper case form.

#Letters ordered by frequency in English words as per
# https://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/
# frequencies.html
NOT_ROMAN_NUMBERS = ["E," "T," "A," "O," "N," "S" ,"R," "H," "D,"\
                     "L," "U," "C,"  "M," "F," "Y," "W," "G," "P,"\
                     "B," "K," "Q," "J," "Z"]
NUMBERS = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#These did not consider ;
PUNCTUATION = ['"', "','" "!," "(," "),"  "[," "],"  ,"," ."," " "]
SHORT_PUNCTUATION = [.","  "?," "!," ":"]

OTHER_SPECIAL_CHARACTERS = ["_," "-," "@," "#," "$," "%," "^,"\
                            "&," "*," "<," ">"]
#Define Default Question Strings
WHAT_IS_NEXT_STEP = "What step comes after: "
```

```
WHAT_IS_FINAL_STEP = "What is the final step?"
ACTIVE_NOTES = "The following notes are active: "
QUESTION_MARK_STRING = "?"

#Flashcard Button Index Constants
FC_LEFT = 0
FC_FLIP = 1
FC RIGHT = 2
FC RETURN = 3

#Flashcard Tuple locations
QUESTION = 0
ANSWER = 1
NOTES = 2

#Main Menu Button Index Constants
SELECT_PDF = 0
OPEN_PDF = 1
FLASHCARD = 2
MULT CHOICE = 3
ORDERING = 4
QUIT APP = 5

QUIZ_BUTTONS_ROW = 5

#Multiple Choice constants
MULTIPLE CHOICE QUANTITY = 4
CHOICE SPOT IN ARRAY = 1
CORRECT ANSWER = 2

#Multiple Choice Button Index Constants
MC LEFT = 0
MC CHECK = 1
MC RIGHT = 2
MC RETURN = 3

#Constant variables to determine the number of steps available during
# the Ordered List Quiz
MAX STEPS = 6
MIN STEPS = 3

#Ordering List Question Text
ORDERING_LIST_QUESTION = "Place the below steps in order by"+\
    "clicking on them in order that they appear in the procedure."

#Ordering List Button Constants
UP_BUTTON = 0
DELETE_BUTTON = 1
DOWN BUTTON = 2

#Ordering List Button Width Constant
BUTTON_WIDTH = 56
```

B. NTT MAIN

```
from NTT GUI import MainMenu
import  tkinter


if __name__ == "__main__":
    root = tkinter.Tk()
    menu = MainMenu(root)
    root.mainloop()
```

C. NTT_GUI

```
from  tkinter import *
from  tkinter import messagebox, filedialog
import os
from NTTUtility import returnParentDirectory, returnFileName, returnOnlyFileName
from NTTDataHandling import changePDFintoDataStructure
from NTTConstants import *
from NTT_FlascardGUI import FlashcardGUI
from NTT_MultipleChoiceGUI import MultipleChoiceGUI
from NTT_OrderingQuizGUI import OrderingGUI

class MainMenu(Frame):
    '''This GUI modified from example of the canonical Tk intro application
       Tkinter 8.5 reference: a GUI for Python, http://infohost.nmt.edu/tcc/
                                help/pubs/tkinter/web/minimal-app.html'''

    ########################################
    # Constructor
    def __init__(self, master=None):
        self.master = master

        # Initialise the base class
        Frame.  init  (self, master)

        # Set the Window Title
        self.master.title("Navy Training Tool")

        self.buttons = []

        # Display the main window with a little bit of padding
        self.grid(padx=10, pady=10)

        self.createWidgets()

        self.dataStructure = []
        self.steps = []
        self.filename = ""

    ########################################
    # Instance Methods

    def createWidgets(self, width = 20, size = 0):
        """Create all the widgets that we need"""
        count = 1
        self.buttons = []

        if size == 0:
            size = width * width

        # Create the  functionality buttons and associate functions with them
        button = Button(self, text="Select \nDocument," command=lambda arg=self: \
                                                        self.  selectPDF  ())
        button.grid(row=1, column=(int(width/2)))
        self.buttons.append(button)

        #Label will be used to display the name of the selected document
        self.docName = StringVar()
        self.documentLabel = Message(self, textvariable=self.docName, width=500)
        self.documentLabel.grid(row=2, column=(int(width / 2)))

        button = Button(self, text="Open \nDocument," command=lambda \
                                            arg=self: self.__openPDF__())
        button.grid(row=3, column=(int(width / 2)))
        self.buttons.append(button)

        self.blankLabelOne = Message(self, ",", width=500)
        self.blankLabelOne.grid(row=QUIZ_BUTTONS_ROW - 1, column=(int(width / 2)))
```

```
        button = Button(self, text="Flashcards\n," command=lambda arg=self: \
                                                self.  generateFlashcardGUI())
        button.grid(row=QUIZ BUTTONS ROW, column=(int(width / 2) - 1))
        self.buttons.append(button)

        button = Button(self, text="  Multiple  \nChoice," command=lambda \
                                arg=self: self.__generateMultipleChoiceGUI())
        button.grid(row=QUIZ BUTTONS ROW, column=(int(width / 2)))
        self.buttons.append(button)

        button = Button(self, text="Ordering \nQuiz," command=lambda arg=self: \
                                            self.__generateOrderingQuizGUI())
        button.grid(row=QUIZ BUTTONS ROW, column=(int(width / 2) + 1))
        self.buttons.append(button)

        self.blankLabelTwo = Message(self, "," width=500)
        self.blankLabelTwo.grid(row=QUIZ_BUTTONS_ROW + 1, column=(int(width / 2)))

        # Create the  "Quit" Button
        button = Button(self, text="    Quit     \nProgram," command=\
                                                self.  quitButton  )
        button.grid(row=7, column=(int(width/2)))
        self.buttons.append(button)

        #Disable conditional buttons
        for i in range(OPEN_PDF, QUIT_APP):
            self.buttons[i]["state"] = DISABLED


    ########################################
    # private/local Methods
    def __selectPDF__(self):
        #This portion is if a user selects a file.
        try:
            self.filename = filedialog.askopenfilename(initialdir="/," \
                            title="Select file," filetypes=(("pdf files," \
                            "*.pdf"),("all files,""*.*")))

            self.filename = self.filename.replace("/," "\\")

            #If an index is chosen instead of a procedure, have the user
            # re-choose the file.
            if ."index" in self.filename:
                messagebox.showinfo("Invalid File Type," "You chose an index"\
                                    + " file. Please select a procedure.")
                self.  selectPDF  ()

            #Set the filename to the document label
            self.docName.set(returnOnlyFileName(self.filename).strip())
            self.dataStructure = changePDFintoDataStructure(self.filename)
            self.steps = self.dataStructure[STEPS]

########
            #Printing the Header and Details
            print("\nThe following is the Header data:")
            for i in range (len(self.dataStructure[HEADER])):
                print(self.dataStructure[HEADER][i])

            for i in range(len(self.dataStructure[DETAILS])):
                if i == 0:
                    print("\nThe following is the Detail data:")

                print(self.dataStructure[DETAILS][i])

            #Print the steps
            self.dataStructure[STEPS][0].printTree(NNT)
########

            # Enable conditional buttons after a PDF has been selected
```

```
            for i in range(OPEN_PDF, QUIT_APP):
                self.buttons[i]["state"] = 'normal'

        #This catches if the user does not select a file.
        except:

                # Set the filename to the document label to blank space
                self.docName.set("")


    def __openPDF__(self):
        if PDF_EXTENSION in self.filename:
                #Store the code directory to restore path
                codeDirectory = os.getcwd()

                #Change the directory to the PDF location and open it
                # Change to PDF directory
                os.chdir(returnParentDirectory(self.filename))

                # Make the command
                openString = "start " + returnFileName(self.filename)

                # Run the command
                os.system(openString)

                #Restore the path to the code directory
                os.chdir(codeDirectory)
        else:
            messagebox.showwarning("Unselected PDF," "Please select a PDF using"+\
                           " the 'Select PDF' button prior to opening it.")

    def __generateFlashcardGUI(self):
        if PDF_EXTENSION in self.filename:
                self.newWindow = Toplevel(self.master)
                self.app = FlashcardGUI(self.newWindow, self.steps)
        else:
            messagebox.showwarning("Unselected PDF," "Please select a PDF using"\
                        +"the 'Select PDF' button prior to using the flashcards.")

    def __generateMultipleChoiceGUI(self):
        if PDF_EXTENSION in self.filename:
            self.newWindow = Toplevel(self.master)
            self.app = MultipleChoiceGUI(self.newWindow, self.steps)
        else:
            messagebox.showwarning("Unselected PDF," "Please select a PDF using"+\
                        " the 'Select PDF' button prior to using the flashcards.")

    def __generateOrderingQuizGUI(self):
        if PDF_EXTENSION in self.filename:
            self.newWindow = Toplevel(self.master)
            self.app = OrderingGUI(self.newWindow, self.steps)
        else:
            messagebox.showwarning("Unselected PDF," "Please select a PDF using the"\
                           + "'Select PDF' button prior to using the ordering quiz.")

    def __quitButton__(self):
        self.master.destroy()
```

D. NTT_FLASHCARD GUI

```
from  tkinter import *
from  tkinter import messagebox, filedialog, ttk
import os
from NTTConstants import *
from NTTTesting import generateFlashcardQuestion
import time
from NTT_GUI_Functions import *

class FlashcardGUI(Frame):
    '''This GUI modified from example of the canonical Tk intro application
       Tkinter 8.5 reference: a GUI for Python, http://infohost.nmt.edu/tcc/
                                help/pubs/tkinter/web/minimal-app.html'''

    ########################################
    # Constructor
    def __init__(self, master=None, stepsIn = []):
        self.master = master
        self.questionBoolean = True

        # Initialise the base class
        Frame.  init  (self, master)

        # Set the Window Title
        self.master.title("Navy Training Tool: Flashcards")
        self.master.geometry('600x250')
        self.buttons = []
        self.steps = stepsIn

        #Variables for the current question, answer, and notes
        self.question = ""
        self.answer = ""
        self.notes = ""

        self.questionArray = []
        self.questionArrayIndex = 0

        #Create variable to track timing
        self.startTime = time.clock()

        #This array will store the time spent answering each question.
        self.timeArray = []

        # Display the main window with a little bit of padding
        self.grid(padx=10, pady=10)

        self.buildQuestionArray()

        # Initialize Question dependent arrays
        for i in range(len(self.questionArray)):
            self.timeArray.append(0)

        self.createWidgets()


    ########################################
    # Instance Methods

    def createWidgets(self, width = 20, size = 0):
        """Create all the widgets that we need"""

        self.questionFrame = Frame(self, borderwidth=5, width=200, height=100)
        self.questionFrame.grid(row = 0, columnspan = 500, sticky = (NW))
        #self.questionFrame.place(relx=.5, rely=.5, anchor="n")

        self.buttonFrame = Frame(self, borderwidth=5, width=400, height=400)
        self.buttonFrame.grid(row = 1, columnspan = 500, sticky = (SW))
        #self.buttonFrame.place(relx=.5, rely=.5, anchor="c")
```

```
        self.progressFrame = Frame(self, borderwidth=5, width=200, height=100)
        self.progressFrame.grid(row = 2, columnspan = 500, sticky = (SW))
        #self.progressFrame.place(relx=.5, rely=.5, anchor="s")

        self.myLabelVar = StringVar()
        self.questionLabel = Message(self.questionFrame, textvariable = \
                                     self.myLabelVar, width = 500)
        self.questionLabel.grid(columnspan=400, row=0)
        #self.questionLabel.size(20)

        #Will be used to show progess of questions
        self.myProgress = StringVar()
        self.progressLabel = Message(self.progressFrame, textvariable = \
                                     self.myProgress, width = 100)
        self.progressLabel.grid(columnspan = 3, row = 0)

        self.myLabelVar.set(self.questionArray[0][QUESTION])

        self.myProgress.set(str(self.questionArrayIndex + 1) + "/" + \
                                    str(len(self.questionArray)))
        self.buttons = []

        # Create the  functionality buttons and associate functions with them
        button = Button(self.buttonFrame, text="<-," command=lambda arg=self:\
                                                    self.__leftAction())
        button.grid(row=0, column=0)
        button["state"] = DISABLED
        self.buttons.append(button)

        button = Button(self.buttonFrame, text=" Answer ," command=lambda \
                                     arg=self: self.__flipAction())
        button.grid(row=0, column=1)
        self.buttons.append(button)

        button = Button(self.buttonFrame, text="->," command=lambda arg=self:\
                                                    self.__rightAction())
        button.grid(row=0, column=2)
        self.buttons.append(button)

        button = Button(self.buttonFrame, text="  Quit  ," command=\
                                     self.__quitButton__)
        button.grid(row=1, column=1)
        self.buttons.append(button)

        self.startTime = time.clock()

    ########################################
    # private/local Methods
    #Swap between the Question and Answer
    def __flipAction(self):
        if self.questionBoolean:
            self.myLabelVar.set(self.questionArray[self.questionArrayIndex][ANSWER])
            self.buttons[FC FLIP]["text"] = "Question"
            self.questionBoolean = False
        else:
            self.myLabelVar.set(self.questionArray[self.questionArrayIndex]\
                                                                [QUESTION])
            self.buttons[FC_FLIP]["text"] = " Answer "
            self.questionBoolean = True

    #Go to the Question prior
    def __leftAction(self):
        # Update the time it took to complete the last question
        updateQuestionTiming(self)

        # Ensure that the Right button has the arrow text
        self.buttons[FC_RIGHT]["text"] = "->"

        self.buttons[FC_FLIP]["text"] = " Answer "
```

```
    if self.questionArrayIndex >= 1:
        self.questionArrayIndex -= 1

        #After moving left, enable going right to the next question
        self.buttons[FC_RIGHT]["state"] = 'normal'

        #Update the Question text
        self.myLabelVar.set(self.questionArray[self.questionArrayIndex]\
                                                                    [QUESTION])
        self.questionBoolean = True

        #Disable left button if there are no more prior questions
        if self.questionArrayIndex == 0:
            self.buttons[FC_LEFT]["state"] = DISABLED

        self.myProgress.set(str(self.questionArrayIndex + 1) + "/" + \
                                        str(len(self.questionArray)))

#Go to the next Question
def __rightAction(self):
    # Update the time it took to complete the last question
    updateQuestionTiming(self)

    self.buttons[FC_FLIP]["text"] = " Answer "

    #If not on the last question, go right one question.
    if self.questionArrayIndex < (len(self.questionArray) - 1):
        self.questionArrayIndex += 1

        #After moving right, enable left button to be able to go back
        self.buttons[FC_LEFT]["state"] = 'normal'

        #Update the Question text
        self.myLabelVar.set(self.questionArray[self.questionArrayIndex]\
                                                                    [QUESTION])
        self.questionBoolean = True

        self.myProgress.set(str(self.questionArrayIndex + 1) + "/" + \
                                        str(len(self.questionArray)))

        if self.questionArrayIndex == (len(self.questionArray) - 1):
            # Change Right Arrow to Results text if the user is on the last
            # question and all questions have been answered
            checkChangeButtonToResults(self)
    #If there are no more questions, show results.
    else:
        showResults(self)

#Builds the question array for the Flashcards
def buildQuestionArray(self):
    question = ""
    answer = ""
    notes = ""

    #Get the first set of step info and store it in the question array
    question, answer, notes, self.lastNodeUsedIndex = \
                                generateFlashcardQuestion(self.steps)
    self.questionArray.append((question, answer, notes))

    #Cycle through the items until there are no more steps left
    while generateFlashcardQuestion(self.steps, self.lastNodeUsedIndex):
        question, answer, notes, self.lastNodeUsedIndex = \
                generateFlashcardQuestion(self.steps, self.lastNodeUsedIndex)

        self.questionArray.append((question, answer, notes))

def __quitButton__(self):
    self.master.destroy()
```

E. NTT_MULTIPLE CHOICE GUI

```
from  tkinter import *
from  tkinter import messagebox, filedialog, ttk
import os
from NTTUtility import returnParentDirectory, returnFileName
from NTTDataHandling import changePDFintoDataStructure
from NTTConstants import *
from NTTTesting import generateFlashcardQuestion, buildMultipleChoiceArray
import time
from NTT_GUI_Functions import *

class MultipleChoiceGUI(Frame):
    '''This GUI modified from example of the canonical Tk intro application
       Tkinter 8.5 reference: a GUI for Python, http://infohost.nmt.edu/tcc/
                              help/pubs/tkinter/web/minimal-app.html'''
    ########################################
    # Constructor
    def __init__(self, master=None, stepsIn = []):
        self.master = master
        self.questionBoolean = True

        # Initialise the base class
        Frame.  init  (self, master)

        # Set the Window Title
        self.master.title("Navy Training Tool: Multiple Choice")
        self.master.geometry('550x450')
        self.choiceButtons = []
        self.buttons = []
        self.choicesStringVariables = []
        self.steps = stepsIn
        self.stepIndex = 0

        #Variables for the current question, answer, and notes
        self.question = ""
        self.answer = ""
        self.notes = ""

        self.questionArray = []
        self.questionArrayIndex = 0

        #Create variable to track timing
        self.startTime = time.clock()

        # Display the main window with a little bit of padding
        self.grid(padx=10, pady=10)

        self.buildQuestionArray()
        self.choicesArray = buildMultipleChoiceArray(self.questionArray)

        #Create two arrays, one for whether Check button is enabled and other
        # for the user chosen answer
        self.checkEnabledArray = []
        self.usersAnswersArray = []

        # True for Correct, False for Incorrect
        self.correctnessArray = []

        #This array will store the time spent answering each question.
        self.timeArray = []

        #Initialize Question dependent arrays
        for i in range(len(self.questionArray)):
            self.checkEnabledArray.append(True)
            self.usersAnswersArray.append(-1)
            self.correctnessArray.append("")
            self.timeArray.append(0)
```

```
        self.createWidgets()

    ########################################
    # Instance Methods
    def createWidgets(self, width = 20, size = 0):
        """Create all the widgets that we need"""
        self.questionFrame = Frame(self, borderwidth=5, width=200, height=100)
        self.questionFrame.grid(row = 0, columnspan = 500, sticky = (NW))

        self.choicesFrame = Frame(self, borderwidth=5, width=500, height=400)
        self.choicesFrame.grid(row=1, columnspan=500, sticky=(SW))

        self.buttonFrame = Frame(self, borderwidth=5, width=400, height=400)
        self.buttonFrame.grid(row = 2, columnspan = 500, sticky = (SW))

        self.progressFrame = Frame(self, borderwidth=5, width=200, height=100)
        self.progressFrame.grid(row = 3, columnspan = 500, sticky = (SW))

        self.myLabelVar = StringVar()
        self.questionLabel = Message(self.questionFrame, textvariable = \
                                            self.myLabelVar, width = 500)
        self.questionLabel.grid(columnspan=400, row=0)

        #Adds a Answer Color key
        self.colorKeyLabel = Message(self.progressFrame, text="*Legend: " + \
                                    "Green is correct, Red is incorrect,"width=500)
        self.colorKeyLabel.grid(columnspan=500, row=0)

        #Will be used to show progess of questions
        self.myProgress = StringVar()
        self.progressLabel = Message(self.progressFrame, textvariable = \
                                            self.myProgress, width = 100)
        self.progressLabel.grid(columnspan = 3, row = 1)

        self.myLabelVar.set(self.questionArray[0][QUESTION])

        self.myProgress.set(str(self.questionArrayIndex + 1) + "/" + \
                                        str(len(self.questionArray)))

        # Define Choice buttons and text
        self.chosenAnswer = IntVar()

        #Create Choice buttons
        for i in range(MULTIPLE_CHOICE_QUANTITY):
            #Make new choice button
            newChoiceButton = Radiobutton(self.choicesFrame, text=\
                    self.choicesArray[0][CHOICE_SPOT_IN_ARRAY][i], value=i, \
                    variable=self.chosenAnswer, wraplength = 400, justify = LEFT)

            #Assign the button its position
            newChoiceButton.grid(row=i, column=0, sticky=(W))

            #Add the choice to the choiceButtons array
            self.choiceButtons.append(newChoiceButton)

        # Create the  functionality buttons and associate functions with them
        button = Button(self.buttonFrame, text="<-," command=lambda arg=self: \
                                                    self.  leftAction())
        button.grid(row=0, column=0)
        button["state"] = DISABLED
        self.buttons.append(button)

        button = Button(self.buttonFrame, text=" Check ," command=lambda \
                                                arg=self: self.  saveAnswer())
        button.grid(row=0, column=1)
        self.buttons.append(button)

        button = Button(self.buttonFrame, text="->," command=lambda arg=self:\
                                                    self.__rightAction())
```

```
        button.grid(row=0, column=2)
        self.buttons.append(button)

        button = Button(self.buttonFrame, text="  Quit  ," command=\
                                                        self.__quitButton__)
        button.grid(row=1, column=1)
        self.buttons.append(button)

        self.startTime = time.clock()

    ########################################
    # private/local Methods
    #Swap between the Question and Answer
    def __saveAnswer(self):
        if self.buttons[MC_CHECK]["text"] == " Check ":
            updateQuestionTiming(self)
            self.buttons[MC_CHECK]["state"] == DISABLED

            usersChoice = self.chosenAnswer.get()

            self.usersAnswersArray[self.questionArrayIndex] = usersChoice
            self.checkEnabledArray[self.questionArrayIndex] = False

            #Sets Check button state and shows answers when Check button
            # is disabled
            self.setCheckandShowAnswers()

            #Update the Check button text if necessary
            checkChangeButtonToResults(self, MC_CHECK)
        else:
            showResults(self)

    #Go to the Question prior
    def __leftAction(self):
        #Update the time it took to complete the last question
        updateQuestionTiming(self)

        # Set the radio option back to the first option
        self.chosenAnswer.set(0)

        #Handle the left operation if the index is above one, aka.
        # not on the first question
        if self.questionArrayIndex >= 1:
            self.questionArrayIndex -= 1

            #After moving left, enable going right to the next question
            self.buttons[MC_RIGHT]["state"] = 'normal'

            #Update the Question text
            self.myLabelVar.set(self.questionArray[self.questionArrayIndex]\
                                                            [QUESTION])
            self.questionBoolean = True

            #Updates the choices
            for i in range(MULTIPLE_CHOICE_QUANTITY):
                self.choiceButtons[i]["text"] = self.choicesArray[\
                            self.questionArrayIndex][CHOICE SPOT IN ARRAY][i]

            #Disable left button if there are no more prior questions
            if self.questionArrayIndex == 0:
                self.buttons[MC_LEFT]["state"] = DISABLED

            self.myProgress.set(str(self.questionArrayIndex + 1) + "/" + \
                                            str(len(self.questionArray)))

        # Sets Check button state and shows answers when Check button
        # is disabled
        self.setCheckandShowAnswers()
```

```
#Go to the next Question
def __rightAction(self):
    # Update the time it took to complete the last question
    updateQuestionTiming(self)

    #Set the radio option back to the first option
    self.chosenAnswer.set(0)

    if self.questionArrayIndex < (len(self.questionArray) - 1):
        self.questionArrayIndex += 1

        #After moving right, enable left button to be able to go back
        self.buttons[MC_LEFT]["state"] = 'normal'

        # Update the Question text
        self.myLabelVar.set(self.questionArray[self.questionArrayIndex]\
                                                                [QUESTION])
        #Updates the choices
        for i in range(MULTIPLE_CHOICE_QUANTITY):
            self.choiceButtons[i]["text"] = self.choicesArray[\
                            self.questionArrayIndex][CHOICE SPOT IN ARRAY][i]

        #Disable the right button if there are no more questions
        if self.questionArrayIndex == (len(self.questionArray) - 1) and \
                                        (-1 not in self.usersAnswersArray):
            self.buttons[MC_RIGHT]["state"] = DISABLED

        self.myProgress.set(str(self.questionArrayIndex + 1) + "/" + \
                                            str(len(self.questionArray)))
    else:
        #Handle unaswered questions
        if -1 in self.usersAnswersArray:
            #If the last question is the only unanswered question,
            # then stay there
            if self.usersAnswersArray.index(-1) == \
                                    len(self.usersAnswersArray) - 1:
                pass
            #Go to unanswered question
            elif -1 in self.usersAnswersArray:
                firstUnansweredQuestionSpot = self.usersAnswersArray.index(-1) + 1
                self.questionArrayIndex = firstUnansweredQuestionSpot
                self.  leftAction()
        else:
            #showResults(self)
            self.buttons[MC_RIGHT]["state"] = DISABLED

    #Sets Check button state and shows answers when Check button is disabled
    self.setCheckandShowAnswers()

#Builds the question array for the Flashcards
def buildQuestionArray(self):
    question = ""
    answer = ""
    notes = ""

    #Get the first set of step info and store it in the question array
    question, answer, notes, self.lastNodeUsedIndex = \
                                        generateFlashcardQuestion(self.steps)
    self.questionArray.append((question, answer, notes))

    #Cycle through the items until there are no more steps left
    while generateFlashcardQuestion(self.steps, self.lastNodeUsedIndex) != False:
        question, answer, notes, self.lastNodeUsedIndex = \
                generateFlashcardQuestion(self.steps, self.lastNodeUsedIndex)

        self.questionArray.append((question, answer, notes))

def __quitButton__(self):
```

```
        self.master.destroy()

    #Checks the checkEnabledArray to set the value of the check button
    def setCheckButtonState(self):
        if self.checkEnabledArray[self.questionArrayIndex] == True or \
                            self.buttons[MC_CHECK]["text"] == "Results":
            self.buttons[MC_CHECK]["state"] = NORMAL
        else:
            self.buttons[MC_CHECK]["state"] = DISABLED

    def showGradedAnswers(self):
        # Handle if the user chose the wrong answer
        if self.usersAnswersArray[self.questionArrayIndex] != \
                    self.choicesArray[self.questionArrayIndex][CORRECT ANSWER]:
            self.showWrongAnswer()
            self.correctnessArray[self.questionArrayIndex] = False
        else:
            self.correctnessArray[self.questionArrayIndex] = True

        # Show the correct answer
        self.showCorrectAnswer()

    def showWrongAnswer(self):
        self.choiceButtons[self.usersAnswersArray[self.questionArrayIndex]]\
                                                                ["bg"] = "RED"

    def showCorrectAnswer(self):
        self.choiceButtons[self.choicesArray[self.questionArrayIndex]\
                                        [CORRECT_ANSWER]]["bg"] = "GREEN"

    def clearAnswerColor(self):
        for i in range(MULTIPLE_CHOICE_QUANTITY):
            self.choiceButtons[i]["bg"] = "SystemButtonFace"

    #This function changes the Check button state and shows Graded
    # answers when the Check button is DISABLED
    def setCheckandShowAnswers(self):
        self.setCheckButtonState()

        if self.buttons[MC_CHECK]["state"] == DISABLED or \
                            self.buttons[MC_CHECK]["text"] == "Results":
            self.clearAnswerColor()
            self.showGradedAnswers()
        else:
            self.clearAnswerColor()
```

F. NTT_ORDERING QUIZ GUI

```
from  tkinter import *
from  tkinter import messagebox, filedialog, ttk
import os
from NTTUtility import returnParentDirectory, returnFileName
from NTTDataHandling import changePDFintoDataStructure
from NTTConstants import *
from NTTTesting import generateFlashcardQuestion, buildMultipleChoiceArray, \
                                        buildOrderedListQuestionsAndAnswers
from NTT_GUI_Functions import *

class OrderingGUI(Frame):
    '''This GUI modified from example of the canonical Tk intro application
       Tkinter 8.5 reference: a GUI for Python, http://infohost.nmt.edu/tcc/
                                help/pubs/tkinter/web/minimal-app.html'''
    ########################################
    # Constructor
    def __init__(self, master=None, stepsIn = []):
        self.master = master
        self.questionBoolean = True

        # Initialise the base class
        Frame.  init  (self, master)

        # Set the Window Title
        self.master.title("Navy Training Tool: Ordering Quiz")
        self.master.geometry('1000x750')
        self.choiceButtons = []
        self.buttons = []
        self.usersOrderedButtons = []
        self.stepMovementButtons = []

        #Counts the number of items the user has selected
        self.orderedCount = 0

        #Indicated which page of choices that the program is on
        self.currentPage = 0

        self.steps = stepsIn
        self.stepIndex = 0

        #Variables for the current question, answer, and notes
        self.question = ""
        self.answer = ""
        self.notes = ""

        self.questionArray = []
        self.questionArrayIndex = 0

        #Create variable to track timing
        self.startTime = time.clock()

        # Display the main window with a little bit of padding
        self.grid(padx=10, pady=10)

        self.buildQuestionArray()
        self.choicesArray = buildMultipleChoiceArray(self.questionArray)
        self.questionList, self.answerList = \
                    buildOrderedListQuestionsAndAnswers(self.steps)

        #Create two arrays, one for whether Check button is enabled
        # and other for the user chosen answer
        self.checkEnabledArray = []
        self.usersAnswersArray = []

        #Tracks if a page has been answered
        self.pageAnswered = []
```

```
        #Array to be used to determine whether an Question was
        # answered correctly.
        # True for Correct, False for Incorrect
        self.correctnessArray = []

        #This array will store the time spent answering each question.
        self.timeArray = []

        #Initialize Question dependent arrays
        #These array are 2 dimensional to account for the page and its options
        for page in range(len(self.questionList)):
            tempEnabledRow = []
            tempUsersAnswerRow = []
            tempCorrectnessRow = []
            tempTimeRow = []

            for items in range(len(self.questionList[page])):
                tempEnabledRow.append(True)
                tempUsersAnswerRow.append(-1)
                tempCorrectnessRow.append("")
                #tempTimeRow.append(0)

            self.checkEnabledArray.append(tempEnabledRow)
            self.usersAnswersArray.append(tempUsersAnswerRow)
            self.correctnessArray.append(tempCorrectnessRow)
            self.timeArray.append(0)      #tempTimeRow)

            self.pageAnswered.append(False)

        self.createWidgets()

    ########################################
    # Instance Methods
    def createWidgets(self, width = 20, size = 0):
        """Create all the widgets that we need"""
        self.questionFrame = Frame(self, borderwidth=5, width=200, height=100)
        self.questionFrame.grid(row = 0, columnspan = 500, sticky = (NW))

        self.choicesFrame = Frame(self, borderwidth=5, width=500, height=400)
        self.choicesFrame.grid(row=1, columnspan=500, sticky=(W))

        self.buttonFrame = Frame(self, borderwidth=5, width=400, height=400)
        self.buttonFrame.grid(row = 2, columnspan = 500, sticky = (SW))

        self.progressFrame = Frame(self, borderwidth=5, width=200, height=100)
        self.progressFrame.grid(row = 3, columnspan = 500, sticky = (SW))

        self.myLabelVar = StringVar()
        self.questionLabel = Message(self.questionFrame, text =\
                                     ORDERING LIST QUESTION, width = 500)
        self.questionLabel.grid(columnspan=400, row=0)

        #Adds a Answer Color key
        self.colorKeyLabel = Message(self.progressFrame, text="*Legend: "+\
                             "Green is correct, Red is incorrect," width=500)
        self.colorKeyLabel.grid(columnspan=500, row=0)

        #Will be used to show progess of questions
        self.myProgress = StringVar()
        self.progressLabel = Message(self.progressFrame, textvariable = \
                                                 self.myProgress, width = 100)
        self.progressLabel.grid(columnspan = 3, row = 1)

        self.myLabelVar.set(self.questionArray[0][QUESTION])
        self.updateProgressLabel()
        self.questionStepsVar = StringVar()
        self.questionStepsVar.set("Step Options")
        self.questionLabel = Message(self.choicesFrame, textvariable = \
                                     self.questionStepsVar, width=400)
```

```
        self.questionLabel.grid(row=0, column=0)

        # Define Choice buttons and text
        self.chosenAnswer = IntVar()

        #Create Choice buttons
        for i in range(len(self.questionList[0])):
            #Make new choice button
            newChoiceButton = Button(self.choicesFrame, \
                        text=self.giveQuestionOrAnswerText(i), command=lambda \
                        arg=(i): self.  updateSelection(arg), width = BUTTON WIDTH,\
                        wraplength=400, justify=LEFT)

            #Assign the button its position
            newChoiceButton.grid(row=i+1, column=0, sticky=(W))

            #Add the choice to the choiceButtons array
            self.choiceButtons.append(newChoiceButton)

        self.chosenStepsVar = StringVar()
        self.chosenStepsVar.set("")
        self.chosenLabel = Message(self.choicesFrame, textvariable=\
                                            self.chosenStepsVar, width=400)
        self.chosenLabel.grid(row=0, column=3)

        # Create the  functionality buttons and associate functions with them
        self.blankLabel = Message(self.buttonFrame, text = ",” width = 500)
        self.blankLabel.grid(row=0, column=0)

        button = Button(self.buttonFrame, text="<-,” command=lambda arg=self: \
                                                    self.__leftAction())
        button.grid(row=1, column=0)
        button["state”] = DISABLED
        self.buttons.append(button)

        button = Button(self.buttonFrame, text=" Check ,” command=lambda \
                                                arg=self: self.  saveAnswer())
        button.grid(row=1, column=1)
        self.buttons.append(button)

        button = Button(self.buttonFrame, text="->,” command=lambda arg=self:\
                                                    self.__rightAction())
        button.grid(row=1, column=2)
        self.buttons.append(button)

        button = Button(self.buttonFrame, text="  Quit  ,” command=\
                                                self.__quitButton__)
        button.grid(row=2, column=1)
        self.buttons.append(button)

        self.startTime = time.clock()

    ########################################
    # private/local Methods
    #Swap between the Question and Answer
    def __saveAnswer(self):
        #Handle the Check option
        if self.buttons[MC_CHECK]["text”] == " Check ":
            #Verify that the user has selected all the choices.
            if len(self.usersOrderedButtons) != len(self.choiceButtons):
                messagebox.showinfo("Incomplete Selection,” "You have not ordered”+\
                        " all the steps.\nPlease continue ordering steps or return.”)
            #Compare the user's selected order with the correct answers
            else:
                updateQuestionTiming(self)
                self.buttons[MC_CHECK]["state”] = DISABLED
                self.stepMovementButtons[UP_BUTTON]["state”] = DISABLED
                self.stepMovementButtons[DELETE_BUTTON]["state”] = DISABLED
                self.stepMovementButtons[DOWN_BUTTON]["state”] = DISABLED
```

```
            self.pageAnswered[self.currentPage] = True

            for answerIndex in range(len(self.answerList[self.currentPage])):
                #Store the user's choice and turn button state to False
                self.usersAnswersArray[self.currentPage][answerIndex] = \
                                    self.usersOrderedButtons[answerIndex]["text"]
                self.checkEnabledArray[self.questionArrayIndex] = False

                # Handle display the full list of steps
                self.choiceButtons[answerIndex]["text"] = \
                    self.giveQuestionOrAnswerText(answerIndex)

            #Handle coloring
            self.setCheckandShowAnswers()

            if self.checkIfAllAnswered():
                self.buttons[MC_CHECK]["state"] = NORMAL
                self.buttons[MC_CHECK]["text"] = "Results"

    #Handles generating the results
    else:
        self.showPageResults()

#Go to the Question prior
def __leftAction(self):
    # Update the time it took to complete the last page
    updateQuestionTiming(self)

    self.currentPage -= 1
    self.checkLeftAndRightBoundaries()
    self.setCheckButtonState()
    self.clearOldButtons()
    self.updateChoiceButtonText()
    self.restoreUserChoices()
    self.updatedChoiceButtonsState()
    self.updateProgressLabel()

#Go to the next Question
def __rightAction(self):
    # Update the time it took to complete the last page
    updateQuestionTiming(self)

    self.currentPage += 1
    self.checkLeftAndRightBoundaries()
    self.setCheckButtonState()
    self.clearOldButtons()
    self.updateChoiceButtonText()
    self.restoreUserChoices()
    self.updatedChoiceButtonsState()
    self.updateProgressLabel()

#Builds the question array for the Flashcards
def buildQuestionArray(self):
    question = ""
    answer = ""
    notes = ""

    #Get the first set of step info and store it
    # in the question array
    question, answer, notes, self.lastNodeUsedIndex = \
                        generateFlashcardQuestion(self.steps)
    self.questionArray.append((question, answer, notes))

    #Cycle through the items until there are no more steps left
    while generateFlashcardQuestion(self.steps, self.lastNodeUsedIndex):
        question, answer, notes, self.lastNodeUsedIndex = \
                generateFlashcardQuestion(self.steps, self.lastNodeUsedIndex)
        self.questionArray.append((question, answer, notes))
```

```
def __quitButton__(self):
    self.master.destroy()

#Checks the pageAnswered array to set the Check button
def setCheckButtonState(self):
    if self.pageAnswered[self.currentPage] == False or \
                    self.buttons[MC_CHECK]["text"] == "Results":
        self.buttons[MC_CHECK]["state"] = NORMAL
    else:
        self.buttons[MC_CHECK]["state"] = DISABLED

def showGradedAnswers(self, indexIn):
    # Handle if the user chose the wrong answer
    if self.usersOrderedButtons[indexIn]["text"] != \
                            self.answerList[self.currentPage][indexIn]:
        self.showWrongAnswer(indexIn)
        self.correctnessArray[self.currentPage][indexIn] = False

        #Show if an answer was wrong
        self.showWrongAnswer(indexIn)
    else:
        self.correctnessArray[self.currentPage][indexIn] = True

        # Show if an answer was correct
        self.showCorrectAnswer(indexIn)

def showWrongAnswer(self, indexIn):
    self.usersOrderedButtons[indexIn]["bg"] = "RED"

def showCorrectAnswer(self, indexIn):
    self.usersOrderedButtons[indexIn]["bg"] = "GREEN"

def clearAnswerColor(self, indexIn):
    self.usersOrderedButtons[indexIn]["bg"] = "SystemButtonFace"

#This function changes the Check button state and shows Graded answers
#  when the Check button is DISABLED
def setCheckandShowAnswers(self):
    self.setCheckButtonState()

    for itemIndex in range(len(self.questionList[self.currentPage])):
        if self.buttons[MC_CHECK]["state"] == DISABLED or \
                        self.buttons[MC_CHECK]["text"] == "Results":
            self.clearAnswerColor(itemIndex)
            self.showGradedAnswers(itemIndex)
        else:
            self.clearAnswerColor(itemIndex)

#This function takes the last selected step and places it into the user's chosen
# steps list
def __updateSelection(self, numberIn):
    self.choiceButtons[numberIn]["state"] = DISABLED
    self.chosenStepsVar.set("Chosen Steps")
    newOrderedButton = Radiobutton(self.choicesFrame, text = \
            self.choiceButtons[numberIn]["text"], value=self.orderedCount,\
            variable=self.chosenAnswer, wraplength = 400, justify = LEFT)

    # Assign the button its position
    newOrderedButton.grid(row=self.orderedCount+1, column=3, sticky=(W))

    #If there were no ordered items yet, the also add choice movement buttons
    if self.usersOrderedButtons == []:
        button = Button(self.choicesFrame, text="   Move step up   ," \
                                    command=lambda arg=self: self.__moveUp())
        button.grid(row=1, column=5)
        button["state"] = DISABLED
        self.stepMovementButtons.append(button)

        button = Button(self.choicesFrame, text="     Delete step     ," \
```

```
                                command=lambda arg=self: self.__deleteChoice())
            button.grid(row=2, column=5)
            self.stepMovementButtons.append(button)

            button = Button(self.choicesFrame, text="Move step down," command=\
                                        lambda arg=self: self.__moveDown())
            button.grid(row=3, column=5)
            button["state"] = DISABLED
            self.stepMovementButtons.append(button)
        else:
            self.stepMovementButtons[UP_BUTTON]["state"] = NORMAL
            self.stepMovementButtons[DOWN_BUTTON]["state"] = NORMAL

        self.usersOrderedButtons.append(newOrderedButton)
        self.orderedCount += 1

    #Move the selected step up one spot
    def __moveUp(self):
        usersChoice = self.chosenAnswer.get()

        #Swap the current choice with the previous one
        if usersChoice > 0:
            tempString = self.usersOrderedButtons[usersChoice - 1]["text"]
            self.usersOrderedButtons[usersChoice - 1]["text"] = \
                                self.usersOrderedButtons[usersChoice]["text"]
            self.usersOrderedButtons[usersChoice]["text"] = tempString

            #Update the radio button marker
            self.chosenAnswer.set(usersChoice - 1)

    # Delete the selected step
    def __deleteChoice(self):
        usersChoice = self.chosenAnswer.get()

        #Enable the appropriate choice button
        chosenOrderedStepText = self.usersOrderedButtons[usersChoice]["text"]

        for i in range(len(self.choiceButtons)):
            if self.choiceButtons[i]["text"] == chosenOrderedStepText and \
                                    self.choiceButtons[i]["state"] == DISABLED:
                self.choiceButtons[i]["state"] = NORMAL
                break

        # Shift the text on the chosen steps buttons
        totalOrderedLength = len(self.usersOrderedButtons)

        for i in range(totalOrderedLength - 1):
            if i >= usersChoice:
                self.usersOrderedButtons[i]["text"] = \
                            self.usersOrderedButtons[i+1]["text"]

        #After shifting the text, delete the last choice button
        self.usersOrderedButtons[totalOrderedLength - 1].destroy()
        self.usersOrderedButtons = self.usersOrderedButtons\
                                        [0:totalOrderedLength - 1]

        #Set the radio button selection to 0 and disable or delete
        # the movement buttons
        shortenedOrderedLength = totalOrderedLength -1

        if (shortenedOrderedLength) >= 1:
            self.chosenAnswer.set(0)

        if (shortenedOrderedLength) <= 1:
            self.stepMovementButtons[UP_BUTTON]["state"] = DISABLED
            self.stepMovementButtons[DOWN_BUTTON]["state"] = DISABLED

        if (shortenedOrderedLength) == 0:
            self.chosenStepsVar.set("")
```

```
            self.stepMovementButtons[DELETE_BUTTON]["state"] = DISABLED
            self.stepMovementButtons[DOWN_BUTTON].destroy()
            self.stepMovementButtons[DELETE_BUTTON].destroy()
            self.stepMovementButtons[UP_BUTTON].destroy()
            self.stepMovementButtons = []

        #Decrement since a choice has been deleted
        self.orderedCount -= 1

    # Move the selected step down one spot
    def __moveDown(self):
        usersChoice = self.chosenAnswer.get()

        #Swap the current choice with the following one
        if usersChoice < (len(self.usersOrderedButtons) - 1):
            tempString = self.usersOrderedButtons[usersChoice + 1]["text"]
            self.usersOrderedButtons[usersChoice + 1]["text"] = \
                                self.usersOrderedButtons[usersChoice]["text"]
            self.usersOrderedButtons[usersChoice]["text"] = tempString

            #Update the radio button marker
            self.chosenAnswer.set(usersChoice + 1)

    #If all questions have been answered, this returns True and
    # False otherwise.
    def checkIfAllAnswered(self):
        if False in self.pageAnswered:
            return False
        else:
            return True

    #Remove any movement buttons from a previous page
    def clearMovementButtons(self):
        #Remove any movement buttons from a previous page
        if self.stepMovementButtons != []:
            self.stepMovementButtons[DOWN_BUTTON].destroy()
            self.stepMovementButtons[DELETE_BUTTON].destroy()
            self.stepMovementButtons[UP_BUTTON].destroy()
            self.stepMovementButtons = []

    #Remove the user's previous ordered steps
    def clearOrderedSteps(self):
        # Clear out old user choices
        orderedButtonCount = len(self.usersOrderedButtons)

        # Remove the choices from a previous page
        for buttonChoice in range(orderedButtonCount):
            differenceIndex = orderedButtonCount - buttonChoice - 1
            self.usersOrderedButtons[differenceIndex].destroy()
            self.usersOrderedButtons = self.usersOrderedButtons[0:differenceIndex]

        self.orderedCount = 0

    def clearOldButtons(self):
        self.clearMovementButtons()
        self.clearOrderedSteps()
        self.chosenStepsVar.set("")

    #Check for left and right boundaries and disable those buttons
    def checkLeftAndRightBoundaries(self):
        # Disable the left button if there are no more previous pages
        if self.currentPage == 0:
            self.buttons[MC_LEFT]["state"] = DISABLED
        else:
            self.buttons[MC_LEFT]["state"] = NORMAL

        # Disable the right button if there are no more pages after
        if self.currentPage == (len(self.questionList) - 1):
            self.buttons[MC_RIGHT]["state"] = DISABLED
```

```
        else:
            self.buttons[MC_RIGHT]["state"] = NORMAL

    #Change the Choice buttons to the correct state
    def updatedChoiceButtonsState(self):
        for buttonIndex in range(len(self.choiceButtons)):
            if self.pageAnswered[self.currentPage] == True:
                self.choiceButtons[buttonIndex]["state"] = DISABLED

                # Display the correct step orders
                self.choiceButtons[buttonIndex]["text"] =
                                        self.answerList[self.currentPage][buttonIndex]
            else:
                self.choiceButtons[buttonIndex]["state"] = NORMAL

    #Update the Choice button's text and quantity
    def updateChoiceButtonText(self):
        oldChoicesCount = len(self.choiceButtons)
        newChoicesCount = len(self.questionList[self.currentPage])

        #Store the qunatity of the smaller number of buttons
        smallerRange = 0

        if oldChoicesCount < newChoicesCount:
            smallerRange = oldChoicesCount
        else:
            smallerRange = newChoicesCount

        #Reset the text of the smallerRange number of buttons
        for i in range(smallerRange):
            self.choiceButtons[i]["text"] = self.giveQuestionOrAnswerText(i)

            if self.pageAnswered[self.currentPage] == True:
                self.choiceButtons[i]["state"] = DISABLED
            else:
                self.choiceButtons[i]["state"] = NORMAL

        #If there are more choices to add, then make new buttons
        if newChoicesCount > oldChoicesCount:
            for i in range(oldChoicesCount, newChoicesCount):
                # Make new choice button
                newChoiceButton = Button(self.choicesFrame, \
                            text=self.giveQuestionOrAnswerText(i), command=lambda\
                            arg=(i): self.  updateSelection(arg), width = \
                            BUTTON WIDTH, wraplength=400, justify=LEFT)

                # Assign the button its position
                newChoiceButton.grid(row=i+1, column=0, sticky=(W))

                # Add the choice to the choiceButtons array
                self.choiceButtons.append(newChoiceButton)
        #Delete any extra buttons
        elif newChoicesCount < oldChoicesCount:
            for i in range(newChoicesCount, oldChoicesCount):
                # After shifting the text, delete the last choice button
                self.choiceButtons[i].destroy()

            self.choiceButtons = self.choiceButtons[0:newChoicesCount]

    #Creates the user's answers radio buttons and colors them
    def restoreUserChoices(self):
        if self.pageAnswered[self.currentPage] == True:
            self.chosenStepsVar.set("Chosen Steps")

            for answerIndex in range(len(self.usersAnswersArray[self.currentPage])):
                newOrderedButton = Radiobutton(self.choicesFrame, \
                            text=self.usersAnswersArray[self.currentPage][answerIndex],\
                            value=self.orderedCount, variable=self.chosenAnswer, \
                            wraplength=400, justify=LEFT)
```

```
            # Assign the button its position
            newOrderedButton.grid(row=self.orderedCount+1, column=3, sticky=(W))
            self.usersOrderedButtons.append(newOrderedButton)
            self.orderedCount += 1

        self.setCheckandShowAnswers()

#This returns the answer if the current page has been answered, or will
# return the question otherwise
def giveQuestionOrAnswerText(self, indexIn):
    if self.pageAnswered[self.currentPage] == True:
        self.questionStepsVar.set("Correct Step Order")
        self.questionLabel["bg"] = "GREEN"

        return self.answerList[self.currentPage][indexIn]
    else:
        self.questionStepsVar.set("Step Options")
        self.questionLabel["bg"] = "SystemButtonFace"
        return self.questionList[self.currentPage][indexIn]

#Updates the label that shows the page progress
def updateProgressLabel(self):
    self.myProgress.set(str(self.currentPage + 1) + "/" + \
                                    str(len(self.answerList)))

# Show time and percentages for questions
def showPageResults(self):
    outputString = ""

    # If there is a correctnessArray, then print results
    correctCount = 0

    totalCount = 0
    pageCorrect = []

    for page in range(len(self.correctnessArray)):
        pageCount = 0
        totalCount += len(self.correctnessArray[page])

        for pageItem in range(len(self.correctnessArray[page])):
            if self.correctnessArray[page][pageItem] == True:
                correctCount += 1
                pageCount += 1

        pageCorrect.append(pageCount)

    outputString = ("You answered %d out of %d correctly, achieving %2.2f"\
                    % (correctCount, totalCount, (correctCount / totalCount * 100)))\
                    + " percent.\n"

    # Handle showing the time each question took
    outputString += "\nThe following is a accuracy and time break down per page:\n"

    for i in range(len(self.usersAnswersArray)):
        outputString += "Page %d: %d out of %d correctly, achieving %2.2f " + \
                        "percent, %5.2f" +\
                        % (i+1, pageCorrect[i], len(self.usersAnswersArray[i]), \
                        (pageCorrect[i] / len(self.usersAnswersArray[i]) * 100), \
                        self.timeArray[i]) +  " seconds\n"

    messagebox.showinfo("Ordering Results," outputString)
```

G. NTT_GUI_FUNCTIONS

```
from  tkinter import *
from  tkinter import messagebox, filedialog, ttk
import time
from NTTConstants import *

# Show time and percentages for questions
def showResults(self):
    outputString = ""

    #If there is a correctnessArray, then print results
    try:
        correctCount = 0

        for i in range(len(self.correctnessArray)):
            if self.correctnessArray[i] == True:
                correctCount += 1

        outputString = ("You answered %d out of %d correctly, achieving %2.2f" + \
                        "percent.\n"\
                        % (correctCount, len(self.correctnessArray),\
                        (correctCount / len(self.correctnessArray) * 100)))
    except:
        pass

    #Handle showing the time each question took
    outputString += "\nThe following is a time break down per question:\n"

    for i in range(len(self.questionArray)):
        outputString += "Question %d: %5.2f seconds\n" % (i + 1, \
                                                          self.timeArray[i])
    messagebox.showinfo("Results," outputString)

def updateQuestionTiming(self):
    timeDifference = time.clock() - self.startTime

    #If there is a choiceButtons array, use it for an if statement
    try:
        if self.buttons[MC_CHECK]["state"] == NORMAL:
            #Used to get time for the Ordering Quiz
            try:
                self.timeArray[self.currentPage] += timeDifference
            #Used to get time for the Multiple Choice
            except:
                self.timeArray[self.questionArrayIndex] += timeDifference
    #Used to get the time for the Flashcards
    except:
        self.timeArray[self.questionArrayIndex] += timeDifference

    self.startTime = time.clock()

# This function will change the Right button text to Results if the user
#  is on the last question and all questions
# have been answered.
def checkChangeButtonToResults(self, buttonToChange = MC_RIGHT):
    #If there is a usersAnswerArray, handle the check
    try:
        # Change Right Arrow to Results text if the user is on the last
        #  question and all questions have been answered
        if (-1 not in self.usersAnswersArray):
            self.buttons[buttonToChange]["text"] = "Results"
            self.buttons[buttonToChange["state"]] = NORMAL
    #If there is no usersAnswerArray, change the right button to Results
    except:
        self.buttons[buttonToChange]["text"] = "Results"
        self.buttons[buttonToChange]["state"] = NORMAL
```

H. NTT TESTING

```
from NTTConstants import *
from NTTUtility import removeStepNumber, isNote, makeIntoQuestions
from StepsNode import StepsNode
from random import randint

#ROOT is a constant for 0, it is the first index of the treeIn list.
def generateFlashcardQuestion(treeIn, lastNodeUsedIndex = ROOT):
    questionOut = ""
    answerOut = ""
    notesOut = ""
    lastNodeUsedOut = lastNodeUsedIndex

    #Handles generating the first flash card from the tree
    if lastNodeUsedIndex == ROOT:
        if treeIn[ROOT].childrenCount() > 0:
            newNode = treeIn[ROOT].children[0]
        else:
            raise "There were no children under the root node to use."

        indexNumber = treeIn.index(newNode)

        #If the current node has children
        if treeIn[indexNumber].childrenCount() > 0:
            questionOut = generateQuestion(newNode)

            #Get the notes, answer, and last node from the
            # buildNotes and Flashcard Answer function
            notesOut, answerOut, lastNodeUsedOut = \
                            buildNotesAndFlashcardAnswer(treeIn, \
                            newNode.children[0])
    #Handles the question and answer generation if a
    # lastNodeUsed was given
    else:
        #Handles if the last node has children
        if treeIn[lastNodeUsedIndex].hasChildren and len(\
                        treeIn[lastNodeUsedIndex].children) > 0:
            questionOut = generateQuestion(treeIn[lastNodeUsedIndex])
            notesOut, answerOut, lastNodeUsedOut = \
                                buildNotesAndFlashcardAnswer(treeIn, \
                                treeIn[lastNodeUsedIndex].children[0])
        #Handle the case with no children, but has siblings
        #Keep searching up in the tree until there is a younger
        # sibling or until a main step is reached.
        #A main step being a step that has Root as its parent.
        else:
            newNode = treeIn[lastNodeUsedIndex]
            #This will store where the last question came from
            questionNode = newNode

            while True:
                if newNode == treeIn[ROOT] or lastNodeUsedOut == \
                                                    len(treeIn) - 1:
                    return False

                currentIndex = treeIn.index(newNode)

                #If the parent is a Root, then handles similiar to first node
                if newNode.parentID() == treeIn[ROOT].children[0].parentID():
                    # Handle the case with no children, but has siblings
                    if newNode.hasYoungerSibling():
                        newNode = treeIn[currentIndex].\
                                            getNextYoungerSiblingNode()
                        currentIndex = treeIn.index(newNode)

                        #Question and Answer generation like initial node
                        if len(treeIn[currentIndex].children) > 0:
                            questionOut = generateQuestion(newNode)
```

```
                            # Get the notes, answer, and last node from the
                            # buildNotes and Flashcard Answer function
                            notesOut, answerOut, lastNodeUsedOut = \
                                        buildNotesAndFlashcardAnswer(treeIn, \
                                        treeIn[currentIndex].children[0])
                        #If the most recent step has no children, then
                        # return False
                        else:
                            return False
                    break
                #Generates question if there is a younger sibling node
                elif newNode.hasYoungerSibling():
                    questionOut = generateQuestion(questionNode)
                    notesOut, answerOut, lastNodeUsedOut = \
                                        buildNotesAndFlashcardAnswer(treeIn, \
                                        treeIn[currentIndex].\
                                        getNextYoungerSiblingNode())
                    break
                else:
                    newNode = newNode.parentID()
    return questionOut, answerOut, notesOut, lastNodeUsedOut

def generateQuestion(nodeIn):
    return WHAT_IS_NEXT_STEP + str(makeIntoQuestions(nodeIn.\
                                                     getNodeText()))

#This function will cycle through a given node collecting the
# notes together, as well as getting the answer.
#Function returns the notes, the answer, and the last node used
def buildNotesAndFlashcardAnswer(treeIn, nodeIn):
    indexNumber = treeIn.index(nodeIn)
    notesOut = nodeIn.notes()
    answerOut = nodeIn.getNodeText()

    #Changes the None step into a more understandable statement
    # about not have any sub steps.
    if answerOut.upper() == "NONE":
        answerOut = "There are no sub steps."

    lastNodeUsedOutIndex = treeIn.index(treeIn[indexNumber])

    return notesOut, answerOut, lastNodeUsedOutIndex

#This function provides the valid options for all multiple
# choice questions.
#It removes all of the highest level steps.
def buildMultipleChoiceArray(questionAnswerArrayIn):
    multChoiceOut = []

    #Cycle through all the Question Answer data
    for i in range(len(questionAnswerArrayIn)):
        #Makes a choices array and assigns the default value as " "
        choices = []

        for j in range(MULTIPLE_CHOICE_QUANTITY):
            choices.append("")

        questionOut = ""
        answerSpot = -1
        questionOut = questionAnswerArrayIn[i][QUESTION]
        strippedAnswer = removeStepNumber(questionAnswerArrayIn[i][ANSWER])

        #Add the answer to the temp Answer array
        preShuffledAnswers = [strippedAnswer]

        #Select the random choices
        while True:
            if len(preShuffledAnswers) == MULTIPLE_CHOICE_QUANTITY:
                break
```

```
            randomNumber = randint(0, len(questionAnswerArrayIn) - 1)
            tempAnswer = removeStepNumber(questionAnswerArrayIn\
                                          [randomNumber][ANSWER])

            if tempAnswer not in preShuffledAnswers and \
                                tempAnswer[:-2] not in questionOut:
                preShuffledAnswers.append(tempAnswer)

        #Create an array that has the index has the value
        # within that spot.
        #We need as many entries as multiple choice options.
        optionSpots = []

        for j in range(MULTIPLE_CHOICE_QUANTITY):
            optionSpots.append(j)

        #Shuffle the answers and store the spot number
        # for the right answer
        for j in range(len(preShuffledAnswers)):
            if len(optionSpots) > 1:
                choiceNumber = randint(0, len(optionSpots) - 1)
            else:
                choiceNumber = 0

            choices[optionSpots[choiceNumber]] = preShuffledAnswers[j]

            if j == 0:
                answerSpot = optionSpots[choiceNumber]

            optionSpots.pop(choiceNumber)

        multChoiceOut.append([questionOut, choices, answerSpot])

    return multChoiceOut

#This function takes in the tree and returns a list of all the steps
def buildOrderingStepsArray(treeIn):
    stepList = []
    currentNode = treeIn[ROOT]
    rootName = currentNode.nodeID

    #Traverse the Step Tree and take the text from each node and append
    # it to the stepList array.
    while True:
        #If there was a child, set the child as the currentNode
        if currentNode.hasChildren():
            currentNode = currentNode.children[0]
        #If there were no children, but there was a younger sibling,
        # then set the younger sibling as the currentNode
        elif currentNode.hasYoungerSibling():
            currentNode = currentNode.getNextYoungerSiblingNode()
        #If there were no children or younger siblings, go up the
        # tree to search for siblings
        else:
            while True:
                currentNode = currentNode.parentID()

                #If there Root node was reached, then there are
                # no more entries, so return the stepList
                if str(currentNode.nodeID) == rootName:
                    return stepList

                #Check for younger sibling, if found set to
                # currentNode and break
                if currentNode.hasYoungerSibling():
                    currentNode = currentNode.getNextYoungerSiblingNode()
                    break

        #Append the currentNode's text to the stepList array.
```

```
        #If there was a none step, change the text.
        if currentNode.stepsText().upper() == "NONE":
            stepList.append("There are no sub steps.")
        else:
            stepList.append(currentNode.stepsText())
    return stepList

#This takes an Ordering Steps Array and removes all the step numbers
def buildStrippedOrderedStepsArray(stepsListIn):
    strippedList = []

    for i in range(len(stepsListIn)):
        strippedList.append(removeStepNumber(stepsListIn[i]))

    return strippedList

#This function takes a stripped ordered list and splits it into pages
def buildOrderedStepsPages(strippedOrderedListIn):
    pagedListOut = []
    fullPageCount = int(len(strippedOrderedListIn) / MAX_STEPS)
    lastPageRemainder = len(strippedOrderedListIn) % MAX_STEPS

    #If there are less than MAX STEPS, then append them all
    # to the pagedListOut array
    if len(strippedOrderedListIn) <= MAX_STEPS:
        pagedListOut.append(strippedOrderedListIn)
    elif lastPageRemainder >= MIN_STEPS or lastPageRemainder == 0:
        #Assign steps to the correct page
        while strippedOrderedListIn:
            singlePage = []

            #If there is still a full page of options or more, just
            # add the MAX STEPS to the single page.
            #Pop the term from the input list when it has been used.
            if len(strippedOrderedListIn) >= MAX_STEPS:
                for i in range(MAX_STEPS):
                    singlePage.append(strippedOrderedListIn[0])
                    strippedOrderedListIn.pop(0)
            #If there are less than the MAX STEPS then add them
            # to the last page
            else:
                for i in range(len(strippedOrderedListIn)):
                    singlePage.append(strippedOrderedListIn[0])
                    strippedOrderedListIn.pop(0)

            pagedListOut.append(singlePage)
    #There would be too few items on the last page, so distribute steps to
    # make the pages more even.
    else:
        remainingFullPages = fullPageCount
        shiftedPages = 0
        averageItems = 0

        for i in range(fullPageCount):
            iOffset = i + 1

            if ((iOffset * MAX_STEPS) + lastPageRemainder / iOffset)\
                                                    >= MIN STEPS:
                remainingFullPages -= iOffset
                shiftedPages = iOffset

                #Get the average items per shortened page
                averageItems = int(((iOffset * MAX STEPS) + lastPageRemainder)\
                                                      / (iOffset + 1))
                break

        #Add all the full pages to the pagedListOut
        for i in range(remainingFullPages):
            singlePage = []
```

```
            for i in range(MAX_STEPS):
                singlePage.append(strippedOrderedListIn[0])
                strippedOrderedListIn.pop(0)

            pagedListOut.append(singlePage)

        # Assign steps to the shortened pages
        while strippedOrderedListIn:
            singlePage = []

            #Populate the shortened pages.
            #If there are extra terms, they should go on earlier pages.
            if shiftedPages > 0 and float(((shiftedPages * MAX_STEPS)+ \
                            lastPageRemainder) / (shiftedPages + 1)) > \
                            averageItems:
                loopVariable = averageItems + 1
            else:
                loopVariable = averageItems

            shiftedPages -= 1

            for i in range(int(loopVariable)):
                singlePage.append(strippedOrderedListIn[0])
                strippedOrderedListIn.pop(0)

            pagedListOut.append(singlePage)
    return pagedListOut

#Combine the build, stripping, and paging the Ordered list
def buildStripPageOrderedList(treeIn):
    listItems = buildOrderingStepsArray(treeIn)
    strippedList = buildStrippedOrderedStepsArray(listItems)
    pagedList = buildOrderedStepsPages(strippedList)

    return pagedList

#Get the paged ordered list. Create this list and
# a shuffled one.
#The shuffled one will be the questions, the sorted
#  will be the answer.
def buildOrderedListQuestionsAndAnswers(treeIn):
    answerList = buildStripPageOrderedList(treeIn)
    questionList = []

    for page in range(len(answerList)):
        pageList = []
        spotList = []

        for i in range(len(answerList[page])):
            spotList.append(i)

        for arrayIndex in range(len(answerList[page])):
            if len(spotList) > 1:
                randomNumber = randint(0, len(spotList) - 1)
            else:
                randomNumber = 0

            pageList.append(answerList[page][spotList[randomNumber]])
            spotList.pop(randomNumber)

        questionList.append(pageList)
    return questionList, answerList
```

APPENDIX C. PARSING CODE

Code available online at https://gitlab.nps.edu/jvmoorin/NavyTrainingTool/tree/master

A. STEPS NODE

```
from NTTConstants import NNT, NT, TABS, SPACES, NEW_LINES, SEQUENCE

class StepsNode:
    #Default constructor
    def __init__(self, nodeName, parentName = None):
        self.__nodeID = nodeName
        self.__children = []
        self.__parentID = parentName
        self.__childrenCount = 0
        self.__stepsText = ""
        self.__notes = []

    @property
    #These are the StepsNode's getter functions
    def nodeID(self):
        return self.__nodeID

    def parentID(self):
        return self.__parentID

    def childrenCount(self):
        return self.__childrenCount

    def stepsText(self):
        return self.__stepsText

    def notes(self):
        return self.__notes

    #This specifcal return the node's text as a string
    def getNodeText(self):
        stringOut = self.stepsText()
        return stringOut

    #This specifcal return the node's notes as a string
    def getNodeNotes(self):
        stringOut = self.notes()
        return stringOut

    #This specifcal return the node's id as a string
    def getNodeID(self):
        stringOut = self.nodeID()
        return stringOut

    #Setter function for children count
    def setChildrenCount(self, countIn):
        self.__childrenCount = countIn

    # Setter function for parent node
    def setParentNode(self, nodeIn):
        self.__parentID = nodeIn

    #This function return the pointer for the node
    def nodeReference(self):
        return self
```

```
#Set the value for the text of a node
def setText(self, textIn):
    self.__stepsText = textIn

#Set the value for the notes of a node
def addNotes(self, textIn):
    self.__notes.append(textIn)

def addNotesArray(self, nodeArrayIn):
    self.__notes = nodeArrayIn

def getParentID(self):
    tempNode = self.parentID()

    if tempNode != None:
        return tempNode.nodeID
    else:
        return None

#This function returns a Boolean depending on if a node has children
def hasChildren(self):
    if self.children == []:
        return False
    else:
        return True

#Return Boolean if the current nodes has siblings
def hasSibling(self):
    if len(self.parentID().children) <= 1:
        return False
    else:
        return True

#Only returns true if there is another sibling node to the "right" of
# the current node
def hasYoungerSibling(self):
    if self.hasSibling():
        numberSiblings = len(self.parentID().children)

        #Offset the number of siblings by to account for index number vs
        # actual count
        if self.parentID().children.index(self) < (numberSiblings - 1):
            return True
        else:
            return False
    else:
        return False

#Return the next sibling node
def getNextYoungerSiblingNode(self):
    if self.hasYoungerSibling():
        currentNodeIndex = self.parentID().children.index(self)
        return self.parentID().children[currentNodeIndex + 1]
    else:
        print("The getNextYoungerSiblingNode function was used when there was"\
                                    +"not any Younger Sibling nodes.")

@property
#Children getter function
def children(self):
    return self.__children

#Function to add child node, also updates the children count and assigns
#  the parent value to the child
```

```
    def add_child(self, nodeIn):
        if nodeIn not in self.__children:
            self.__children.append(nodeIn)
            self.setChildrenCount(self.childrenCount() + 1)

        nodeIn.setParentNode(self)

    #Defines the print function for the node
    def __str__(self):
        return 'Steps Node[Name: ' + str(self.nodeID) + ', Parent: ' +\
               str(self.getParentID()) + ', Children: ' + \
               self.printChildrenIDs() + ', Children Count: ' +\
               str(self.childrenCount()) + ', Text: ' + \
               str(self.stepsText()) + ', Notes: ' +\
               str(self.buildNotesString()) + ']'

    #Print the id's for the parent nodes
    def printChildrenIDs(self):
        myChildren = self.children
        outString = ""

        for i in range(self.childrenCount()):
            outString += myChildren[i].nodeID

            if i < (self.childrenCount() - 1):
                outString += ," "
        return outString

    def buildNotesString(self, count = 0, style = SEQUENCE, \
                                              newLineBool = False):
        outString = ""
        newLineCount = 0

        if newLineBool:
            newLineCount = 1

        for i in range(len(self.notes())):
            if style == SPACES:
                outString += count * " " + self.notes()[i] + newLineCount * "\n"
            elif style == TABS:
                outString += count * "\t" + self.notes()[i] + newLineCount * "\n"
            elif style == NEW_LINES:
                outString += self.notes()[i] + count * "\n"
            elif style == SEQUENCE:
                if outString == "":
                    outString = self.notes()[i]
                else:
                    outString += ," " + self.notes()[i]
            else:
                raise "There was an invalid style type used for StepsNode's"+\
                                                  "buildNotesString function."
        return outString

### Can update to print different things
    #This build the tree string
    def buildTreeString(self, style = None, count = 0):
        outString = ""
        tempString = ""

        if self.childrenCount() != []:
            if style == NNT:
                tempString = "\n" + self.buildNotesString(count, TABS, True) +\
                                    (count * "\t") + self.nodeID + " " +\
                                    self.stepsText()
                outString += tempString
```

```
        elif style == NT:
            tempString = self.nodeID() + " " + self.stepsText()
            outString += (count * "\t") + tempString
        else:
            tempString = str(self)
            outString += (count * "\t") + tempString

        for i in range(len(self.children)):
            outString += self.children[i].buildTreeString(style, count + 1)

        return outString
    else:
        if style == NNT:
            tempString = "\n" + self.buildNotesString(count, TABS, True) +\
                                (count * "\t") + self.nodeID() + " "  + \
                                 self.stepsText()
            outString += tempString
        elif style == NT:
            tempString = self.nodeID() + " " + self.stepsText()
            outString += (count * "\t") + tempString
        else:
            tempString = str(self)
            outString += (count * "\t") + tempString

        return outString

#This function prints the tree string after building it
def printTree(self, style = None):
    print(self.buildTreeString(style))
```

B. NTT DATA HANDLING

```
import os
from NTT_HTML_Functions import *
from StepsNode import *

#This function turns a pdf into a text file with html tags
def convertPDFtoHTML(pathIn):
    #This command prints out the Parent directory
    currentDir = returnParentDirectory(pathIn)
    shortName = returnOnlyFileName(pathIn)

    #Define file names
    pdfName = '"' + currentDir + "\\" + shortName + PDF_EXTENSION + '"'
    htmlName = '"' + currentDir + "\\" + shortName + HTML_EXTENSION + '"'
    txtName = shortName + TXT_EXTENSION

    #Make the command string that uses the mutool to turn a pdf
    # into a html file.
    #Command is in the form of "mutool convert -o outputHTMLPath
    # inputPDFPath"
    muCommand = "mutool convert -o " + htmlName + " " + pdfName
    print(muCommand)
    os.system(muCommand)

    #Rename html file so that it is now a text file
    #Command is in the form of "rename originalFilePath outputFileName"
    renameCommand = "rename " + htmlName + " " + txtName
    os.system(renameCommand)
    print(renameCommand)

#Return the contents of a text file
def loadTxtFile(pathIn):
    #This command prints out the Parent directory
    currentDir = returnParentDirectory(pathIn)
    shortName = returnOnlyFileName(pathIn)
    txtName = currentDir + "\\" + shortName + TXT_EXTENSION

    readFile = open(txtName)
    fileContents = readFile.read()

    #This will display the raw text extract from the file.
    #print(fileContents)

    readFile.close()

    #Remove text file to prevent conflicts for future runs and
    # additional HTML files
    deleteFile(txtName)

    return(fileContents)

#This function deletes a given file using Windows Command
# Line "del" command
def deleteFile(fileNameIn):
    deleteFileCommand = "del " + '"' + fileNameIn + '"'
    print(deleteFileCommand)
    os.system(deleteFileCommand)

#This function takes in an array tuples and converts it into
# one nested array.
#Element one of the array will have Header fields, while Element
# two will have the steps.
#The subsequent steps will be further nested under the parent ones.
def generateDataStructure(tupleArrayIn):
    inHeader = True
    lastHeight = -1
    currentHeight = lastHeight
    lastLeft = -1
```

```
currentLeft = lastLeft

#Array that will be used to keep track of bolded header locations
#This will particularly be used to handle multiple bolded terms on
# the same row (sequential).
newestIndices = []

#Used to keep track of the next index to use for bolded entries
# in the Header section
nextIndex = -1

#This will determine which element in the newestIndices list to use
useIndex = -1

#These will indicate the range of indices to add contents to
lowerIndex = -1
upperIndex = -1

#This used to determine stacked or in line bolded terms
lastTermBolded = False

#Arrays for step logic
#This will be used to store left values for the steps
stepSpacing = []

#This will store the current step index in the nested arrays
stepTracker = []

#This will keep trackof the last node at each different level in the tree
nodeArray = []

#Make root node for the steps
rootNode = StepsNode("ROOT")

#Make parent node and current node for future use
parentNode = rootNode
currentNode = rootNode

myTreeNodesArray = [rootNode]

#Create variable to handle NOTES in the steps processing
noteBool = False
noteText = ""
tempNoteArray = []

#Boolean to keep track of entering Detail section
haveDetails = False

subStepCount = -1

#Boolean to keep track of entering Steps section
inSteps = False

#This array will be the final output when all header and steps are
# in the proper place
structuredArray = [[], [], myTreeNodesArray]

#Details Count, used to count number of detail headers
detailCount = -1

detailsPunctuation = False

#Cycle through all elements of tupleArrayIn
#Each element represented a page from the source document
for page in range(len(tupleArrayIn)):
    #Handle the initial Header fields and text
    while inHeader:
        #Permits breaking out of loop to go to the next page for more entries
        if len(tupleArrayIn[page]) == 0:
```

```
        break

    currentHeight = tupleArrayIn[page][0][HTML_TOP]
    currentText = tupleArrayIn[page][0][HTML_TEXT].strip()

    #This breaks out of the loop when a primary step is detected
    if (hasStepNumber(currentText)) and (tupleArrayIn[page][0][HTML_BOLD]\
                                                              == False):
        #Turn boolean False to exit the Header while loop
        inHeader = False

        structuredArray[HEADER] = deleteEmpytHeaderFields(\
                                                structuredArray[HEADER])
        lastTermBolded = False
        inSteps = True
        lastHeight = -1
        break
    #Searches for a Detail
    elif (isAllCaps(currentText)) and (tupleArrayIn[page][0][HTML_BOLD] ==
                                                                  False):
        #If this is a proper detail, there will be a step following it
        if len(tupleArrayIn[page]) > 2 and \
                                hasStepNumber(tupleArrayIn[page][1][HTML_TEXT]):
            # Turn boolean False to exit the Header while loop
            inHeader = False
            structuredArray[HEADER] = deleteEmpytHeaderFields\
                                                    (structuredArray[HEADER])
            lastTermBolded = False

            #Enable the Detials while loop and data processing
            haveDetails = True
            lastHeight = -1
            break
    #The bold handles the Header Field Titles
    if tupleArrayIn[page][0][HTML_BOLD] == True:
        #Deals with sequential Header Field Titles
        if (currentHeight == lastHeight or lastHeight == -1 or \
                                                  lastTermBolded == False):
            #Add items to list, Header items of form [Header Title, Contents]
            structuredArray[HEADER].append([currentText, ""])

            #Reset the indices of interest for the start on new bold terms
            if lastTermBolded == False:
                newestIndices = []
                useIndex = -1

            lastTermBolded = True
            nextIndex += 1
            newestIndices.append(nextIndex)
        #Deals with stacked Header Field Title
        elif lastTermBolded == True:
            structuredArray[HEADER][nextIndex][HEADER_FIELD_NAME] += \
                                                    " " + currentText
    #This deals with the Header Field contents
    else:
        lastTermBolded = False

        #Set upper and lower bounds of indices found in the newestIndices list
        lowerIndex = newestIndices[0]
        upperIndex = newestIndices[-1]

        #Set useIndex if it is still default of -1
        if useIndex == -1:
            useIndex = lowerIndex

        # Get current tuple's left value
        currentLeft = tupleArrayIn[page][0][HTML_LEFT]

        # Set last left value for first time
```

```
            if lastLeft == -1:
                lastLeft = currentLeft
            #Increment if the left are different
            #This indicates a different column/header field
            if currentLeft != lastLeft:
                useIndex += 1
            #Reset useIndex if it exceeds the upper limit
            if useIndex > upperIndex:
                useIndex = lowerIndex

            structuredArray[HEADER][useIndex][HEADER_CONTENT] += currentText + " "

            #Use the currentLeft as next tuple's lastLeft
            lastLeft = currentLeft

        #Remove the last node used
        tupleArrayIn[page].pop(0)

        lastHeight = currentHeight

    #Now process details if they are present
    while haveDetails:
        detailsPunctuation = False

        # Permits breaking out of loop to go to the next page for more entries
        if len(tupleArrayIn[page]) == 0:
            break

        currentText = tupleArrayIn[page][0][HTML TEXT].strip()

        #See if there is any punctuation in the the given text.
        for i in range(len(SHORT_PUNCTUATION)):
            if SHORT_PUNCTUATION[i] in currentText:
                detailsPunctuation = True
                break
        #Determine that there is a step or a note
        if(hasStepNumber(currentText) and isAllCaps(currentText)) or \
                                                    isNote(currentText):
            haveDetails = False
            inSteps = True
            structuredArray[DETAILS] = deleteEmpytDetailsFields\
                                                (structuredArray[DETAILS])
            break
        #Verifies that text is a Detail Header by being ALL CAPS
        # with no punctuation.
        elif isAllCaps(currentText) and not detailsPunctuation:
            detailCount += 1
            structuredArray[DETAILS].append([])
            structuredArray[DETAILS][detailCount].append(currentText)
            structuredArray[DETAILS][detailCount].append([])
            subStepCount = -1
            subStepBool = False
        elif hasStepNumber(currentText):
            detailsPunctuation = False
            subStepCount += 1
            subStepBool = True
            structuredArray[DETAILS][detailCount][1].append(currentText)
        else:
            if subStepBool == True:
                structuredArray[DETAILS][detailCount][1][subStepCount] += " " +\
                                                                currentText
            else:
                structuredArray[DETAILS][detailCount][0] += " " + currentText

        # Remove the last node used
        tupleArrayIn[page].pop(0)

    # Now time to process the steps
    while inSteps:
```

```
# Permits breaking out of loop to go to the next page for more entries
if len(tupleArrayIn[page]) == 0:
    break

currentText = tupleArrayIn[page][0][HTML TEXT].strip()

#Handle any new header fields that come along after having started
# assessing the steps.
#We would expect this at the type and bottom of new pages.
#We check the left value since the top header should always be to the
#  left of any steps below.
#This text will be bolded and in ALL CAPS.
if (tupleArrayIn[page][0][HTML_BOLD] == True) and isAllCaps(currentText)\
                    and stepSpacing[0] > tupleArrayIn[page][0][HTML_LEFT]:
    lastTermBolded = True
    lastHeight = -1
    #Pop to ignore this entry
    tupleArrayIn[page].pop(0)
    continue
#Handle the comments for the bold headers
#Verify that the item has no step number
elif (lastTermBolded == True):
    #Catches if there is a new step after a header item
    if(hasStepNumber(currentText)):
        lastTermBolded = False
    else:
        #Keeps track of the height value to group multiple lines
        # of header info together.
        if lastHeight == -1:
            lastHeight = tupleArrayIn[page][0][HTML_TOP]
        elif (tupleArrayIn[page][0][HTML_TOP] - lastHeight) > 12:
            lastTermBolded = False
        else:
            lastHeight = tupleArrayIn[page][0][HTML TOP]

        # Pop to ignore this entry
        tupleArrayIn[page].pop(0)
        continue

#Check for the leading step number identifier
if (hasStepNumber(currentText)):
    #Get left value to compare for hiherarchy of steps
    currentLeft = tupleArrayIn[page][0][HTML LEFT]
    builtName = ""

    #If there are no steps yet, add information for the first one
    if stepTracker == []:
        stepTracker = [1]
        stepSpacing = [currentLeft]
        builtName = "1"
        leftIndex = -1
    #If the left value matches, then steps of that level have
    # already been used.
    #Grab the appropriate header name
    elif (checkLeftValueInArray(currentLeft, stepSpacing) != False) or\
                                                    (-1 in stepSpacing):
        if (checkLeftValueInArray(currentLeft, stepSpacing) == False):
            stepSpacing[stepSpacing.index(-1)] = currentLeft
        else:
            currentLeft = checkLeftValueInArray(currentLeft, \
                                                    stepSpacing)
        leftIndex = stepSpacing.index(currentLeft)
        nodeNameTuple = getNextNodeName(stepTracker[:(leftIndex + 1)])

        #Extract builtName and update stepTracker values from getNextNodeName
        #  return tuples.
        builtName = nodeNameTuple[0]
        stepTracker = nodeNameTuple[1]
```

```
            #Trim arrays to drop off older steps
            #This is done to clear out old child info
            stepSpacing = stepSpacing[:(leftIndex + 1)]
            nodeArray = nodeArray[:(leftIndex)]
        #If the left value in not found, then there is a new step level
        #elif currentLeft not in stepSpacing:
        elif checkLeftValueInArray(currentLeft, stepSpacing) == False:
            #Adds in the new left vale
            currentLeft = tupleArrayIn[page][0][HTML LEFT]
            stepSpacing.append(currentLeft)

            #Adds in the new beginning index for steps
            #Set to 0, so that getNextNodeName will increment up to 1
            stepTracker.append(0)

            leftIndex = stepSpacing.index(currentLeft)
            nodeNameTuple = getNextNodeName(stepTracker[:(leftIndex + 1)])

            #Extract builtName and update stepTracker values from
            # getNextNodeName return tuples
            builtName = nodeNameTuple[0]
            stepTracker = nodeNameTuple[1]

        parentNode = getParentNode(nodeArray, stepSpacing, currentLeft, rootNode)

        #Create node to add to tree structure
        currentNode = StepsNode(builtName, parentNode)
        currentNode.setText(currentText)

        #Node/Tree setup
        parentNode.add_child(currentNode)

        if currentNode not in myTreeNodesArray:
            myTreeNodesArray.append(currentNode)
        if noteBool:
            # Added to ensure that the last note is associated with its step
            if noteText != "":
                tempNoteArray.append(noteText)

            currentNode.addNotesArray(tempNoteArray)
            noteBool = False
            noteText = ""
            tempNoteArray = []

        nodeArray.append(currentNode)
    #Handle Note items
    elif (isNote(currentText)):
        #This is to enable notes to be under the last node
        if noteBool == False:
            parentNode = currentNode

            # Adds in the new left vale
            stepSpacing.append(-1)

            # Adds in the new beginning index for steps
            # Set to 0, so that getNextNodeName will increment up to 1
            stepTracker.append(0)

            leftIndex = stepSpacing.index(-1)
            noteText = ""
        #If there was a prior note, add the old note to the Note Array and
        # reset the note string
        else:
            tempNoteArray.append(noteText)
            noteText = ""

        noteBool = True
        noteText += currentText
    #Handle single line item None
```

```
            elif (currentText.upper() == "NONE"):
                # Get left value to compare for hierarchy of steps
                currentLeft = tupleArrayIn[page][0][HTML LEFT]

                # Adds in the new left vale
                stepSpacing.append(currentLeft)

                # Adds in the new beginning index for steps
                # Set to 0, so that getNextNodeName will increment up to 1
                stepTracker.append(0)

                leftIndex = stepSpacing.index(currentLeft)
                nodeNameTuple = getNextNodeName(stepTracker[:(leftIndex + 1)])

                # Extract builtName and update stepTracker values from
                # getNextNodeName return tuples
                builtName = nodeNameTuple[0]
                stepTracker = nodeNameTuple[1]

                parentNode = getParentNode(nodeArray, stepSpacing, currentLeft, rootNode)

                # Create node to add to tree structure
                currentNode = StepsNode(builtName, parentNode)

                currentNode.setText(currentText)

                ##Update for node/tree
                parentNode.add child(currentNode)

                if currentNode not in myTreeNodesArray:
                    myTreeNodesArray.append(currentNode)

                nodeArray.append(currentNode)
            #Handle multi-line text
            else:
                #This consolidates the notes together into one line
                if noteBool:
                    noteText += " " + currentText
                #Consolidates the node's text together
                else:
                    currentNode.setText(currentNode.stepsText() + " " + currentText)

            tupleArrayIn[page].pop(0)
    return structuredArray

#This removes any Header fields that have no content.
#The arrayIn is the array of Header fields and their text (2 dimensional array)
#Looks for lack of content text
def deleteEmpytHeaderFields(arrayIn):
    count = 0

    while True:
        if count >= len(arrayIn):
            break
        if (arrayIn != []):
            if arrayIn[count][HEADER_CONTENT] == "":
                arrayIn.pop(count)
            else:
                count += 1
    return arrayIn

#This removes any Detail fields that have no content.
#The arrayIn is the array of Detail fields and their text (2 dimensional array)
#Looks for empyt array of substeps
def deleteEmpytDetailsFields(arrayIn):
    count = 0

    while True:
        if count >= len(arrayIn):
```

```
            break
        if (arrayIn != []):
            if arrayIn[count][HEADER_CONTENT] == []:
                arrayIn.pop(count)
            else:
                count += 1
    return arrayIn

#This function will return a tuple of a string that is composed of
# numbers separated by .”“ and the updated Array.
#Each number represents where the item is in the tree
#A single number will be the children of the root
#This function will also update the last number used in the index array.
def getNextNodeName(arrayIn):
    stringOut = “”

    for i in range(len(arrayIn)):
        if i == (len(arrayIn) - 1):
            arrayIn[i] += 1
            stringOut += str(arrayIn[i])
        else:
            stringOut += str(arrayIn[i]) + .”“
    return stringOut, arrayIn

#This node will look for an index of the current left value inside
# of the Space Tracking (leftArrayIn) array.
#This index will determine the current level of the node.
#Go up one level to find the parent.
def getParentNode(nodeArrayIn, leftArrayIn, currentLeftValue, rootNodeIn):
    indexNumber = leftArrayIn.index(currentLeftValue)

    if indexNumber > 0:
        return nodeArrayIn[indexNumber - 1]
    else:
        return rootNodeIn

#Compare two left values and return a boolean value depending on equality
def checkLeftValue(newLeft, oldLeft, tolerance = 1):
    if newLeft == oldLeft:
        return True
    elif newLeft > oldLeft and newLeft <= oldLeft + tolerance:
        return True
    elif newLeft < oldLeft and newLeft >= oldLeft - tolerance:
        return True
    else:
        return False

#This will take in a left value, an array of left values, and a tolerance
#If the value is in the array, return the left value
#Else return false
def checkLeftValueInArray(newLeft, leftArrayIn, tolerance = 1):
    for i in range(len(leftArrayIn)):
        if checkLeftValue(newLeft, leftArrayIn[i]):
            return leftArrayIn[i]
    return False

#This functions consolidates all the data conversion processes:
#a pdf is converted to html, the html is loaded, the html is converted
#into tuples, and the tuples are parsed into a data structure.
def changePDFintoDataStructure(myDocument):
    convertPDFtoHTML(myDocument)
    htmlContents = loadTxtFile(myDocument)

#####Shows the text after having broken the text into HTML pages and rows
    #print(htmlContents)

    htmlTuples = parseHTMLtextIntoTuples(htmlContents)

    #Remove an tuples that have spaces for text
```

```
    htmlTuples = deleteNoTextTuples(htmlTuples)

#####Prints off the text tuples extracted from the HTML data
    #print(htmlTuples)

    completedDataStructure = generateDataStructure(htmlTuples)
    return completedDataStructure

#Delete the tuples that had no text
def deleteNoTextTuples(tuplesIn):
    outTuples = []

    for page in range(len(tuplesIn)):
        tupleArray = []

        for tupleNumber in range(len(tuplesIn[page])):
            if tuplesIn[page][tupleNumber][HTML_TEXT].strip() != "":
                tupleArray.append(tuplesIn[page][tupleNumber])

        outTuples.append(tupleArray)
    return outTuples
```

C. NTT_HTML_FUNCTIONS

```
from NTTUtility import *
from NTTConstants import *

#This function takes a HTML string and only return the contents
# inside the div.
#Returns array of div contents. Each element is a page of data
# from the source document.
def getDivContents(stringIn):
    divBodyArray = []

    #Get contentsw after the open div tag
    tempArray = stringIn.split("<div style=")

    #Get contents after the open div tags
    for i in range(len(tempArray)):
        #Ignore the header information
        if i == 0:
            continue

        divBodyArray.append(tempArray[i])

    for pageCount in range(len(divBodyArray)):
        #Now trim the remaining portion of the open div tag away
        for i in range(len(divBodyArray[pageCount])):
            if divBodyArray[pageCount][i] == ">":
                divBodyArray[pageCount] = divBodyArray[pageCount][(i+1):]
                break
            elif divBodyArray[pageCount][i] == "<":
                divBodyArray[pageCount] = divBodyArray[pageCount][(i):]
                break

        #Get contents before the close div tag
        tempArray = divBodyArray[pageCount].split("</div>")
        divBodyArray[pageCount] = tempArray[0]
    return divBodyArray

#This takes a block of text and divides it by the HTML tag </p>
def splitIntoRows(blockIn):
    rowItems = blockIn.split("</p>")

    if (rowItems[-1] == "" or rowItems[-1] == " " or rowItems[-1] == "\n"):
        rowItems = rowItems[:-1]
    return rowItems

#Function returns array of tuples for row of HTML data
def makeHTMLTuple(rowIn):
    extraTags = False
    newEntry = ""
    entryArray = []
    boldArray = []

    #This if checks to see if there is bolded text following non-bold text
    if (checkBAfterA(rowIn, "<b>," "</span>")):
        #Use the consolidateRowText to eliminate the bold portion of text
        #  and group it all together
        rowIn = consolidateRowText(rowIn)
    #This checks to see if there are more sets of tags after an initial set
    elif(checkBAfterA(rowIn, "</span>," "</span>")):
        extraTags = True

    #Cut text after the </span> tag
    rowData = rowIn.split('</span>')

    #This portion will take extra text sections and them to an array.
    #Later these array items will be given their own tuples
    if extraTags == True:
        #The first portion of the rowData will be handled below, but
```

```
        # look at the rest of the data.
        spanSplitData = rowData[1:]

        #Cycle through spanSplitData and extract the text from it
        for i in range(len(spanSplitData)):
            #if i % 2 != 0:
            if 'pt;">' in spanSplitData[i]:
                #Get the actual text.
                newEntry = extractTextBetween(spanSplitData[i], 'pt;">', "<")

                #Restore any Unicode identifiers back to normal characters
                newEntry = restoreUnicode(newEntry)

                entryArray.append(newEntry)
            #Keep track of whether the entry was bold or not
            if "<b>" in spanSplitData[i]:
                boldArray.append(True)
            else:
                boldArray.append(False)

    #Cut the string after the top: text
    rowData = rowData[0].split('<p style="top:')

    topString = ""
    leftString = ""
    boldBoolean = False
    textString = ""

    outSet = ["p," "t," ";," ">," "<," '"']

    #Get the top field's number
    for i in range(len(rowData[1])):
        if rowData[1][i] != "p":
            topString += rowData[1][i]
        else:
            break
    #Cut the string after the left: text
    rowData = rowData[1].split("left:")
    #Get the left field's number and the bold value
    for i in range(len(rowData[1])):
        if rowData[1][i] not in outSet:
            leftString += rowData[1][i]
        #Determine if the upcoming text is bold
        elif rowData[1][i] == "<":
            if rowData[1][i+1] == "b":
                boldBoolean = True
            break

    #Cut string after the spay style open tag
    rowData = rowData[1].split("<span style=")
    #Cut string after ;"> to get the final text
    textString = rowData[1].split(';">')[1]
    #Return unicode symbols back to "normal" ones
    textString = restoreUnicode(textString)
    #Change the top and left strings into Ints
    topInt = int(topString)
    leftInt = int(leftString)

    outTuple = (topInt, leftInt, boldBoolean, textString)
    outArray = []
    outArray.append(outTuple)

    if extraTags:
        for i in range(len(entryArray)):
            extraTuple = (topInt, leftInt, boldArray[i], entryArray[i])
            outArray.append(extraTuple)
    return outArray

#This functions takes a block of text and returns an array with
```

```
# html tuple info
def parseHTMLintoTuples(blockIn):
    htmlRows = splitIntoRows(blockIn)
    htmlTuples = []
    for i in range(len(htmlRows)):
        tempArray = makeHTMLTuple(htmlRows[i])
        for i in range(len(tempArray)):
            htmlTuples.append(tempArray[i])
    #This delets the Page   of   line
    htmlTuples = removeEndOfPage(htmlTuples)
    return htmlTuples

#This function takes an array and parsers it into html tuples
def parseHTMLtextIntoTuples(textIn):
    allTuples = []
    pageArray = []
    #Split text into pages based on closing div tag
    pageArray = textIn.split("</div>")
    #Remove trailing tags after the final closing div tag
    pageArray = pageArray[:-1]
    for i in range(len(pageArray)):
        allTuples.append(parseHTMLintoTuples(pageArray[i]))
    return allTuples

#This function will check for an end of page line.
#Specifically it looks for PAGE   OF  , then it will search
#backwards for all items on the same line as that text and
# remove from the list
def removeEndOfPage(pageIn):
    countItems = 0
    finalTopValue = 0

    if ("PAGE" in pageIn[-1][HTML_TEXT] and "OF" in pageIn[-1][HTML_TEXT]):
        #Then do logic to remove the last line of the document
        #This will have to look at the same top values
        countItems = 1

        finalTopValue = pageIn[-1][HTML TOP]

        #Go backwards through items looking for all items on the last line
        while(True):
            if (finalTopValue == pageIn[-(countItems + 1)][HTML_TOP]):
                countItems += 1
            else:
                break

        #This cuts off the end of page items
        pageIn = pageIn[:-countItems]
    return pageIn

#This function will only keep the first HTML tag and through out the rest.
#All the actual text will be put together
#Function used for when bold text is mixed in with plain text
def consolidateRowText(stringIn):
    spanString = "</span>"
    priorToText = 'pt;">'
    myStringParts = stringIn.split(spanString)
    outString = myStringParts[0]

    for i in range(1, len(myStringParts)):
        furtherSplitParts = myStringParts[i].split(priorToText)

        if len(furtherSplitParts) > 1:
            outString += furtherSplitParts[1]

    outString += spanString
    return outString
```

D. NTT UTILITY

```
from NTTConstants import *

#This function removes any file postfixes
def removeFileExtension(nameIn):
    if ."" in nameIn:
        tempArray = nameIn.split(."")
        return tempArray[0]
    else:
        return nameIn

#Takes a given path and return the last folder in the path
def returnParentDirectory(pathIn):
    if ."" in pathIn:
        if "/" in pathIn:
            pathIn = pathIn.replace("/," "\\")

        tempArray = pathIn.split("\\")
        shortenedPath = ""

        for i in range(len(tempArray)-1):
            shortenedPath += tempArray[i] + "\\"

        shortenedPath = shortenedPath[:-1]
        return shortenedPath
    else:
        raise RuntimeError("The NTT Utility returnParentDirectory\
                function was not given a file extension in its path.")

#This returns only the base file name. It throws away its path.
def returnFileName(pathIn):
    if ."" in pathIn:
        tempArray = pathIn.split("\\")
        return tempArray[-1]
    else:
        raise RuntimeError("The NTT Utility returnFileName function \
                        was not given a file extension in its path.")

#This function uses both returnFileName (removes path) and
# removeFileExtension (removes extension) to
#return the base name of the file.
def returnOnlyFileName(pathIn):
    return removeFileExtension(returnFileName(pathIn))

#Verifies that an input is a number
def checkNumeric(value):
    if not isinstance(value, int) and not isinstance(value, float):
        raise RuntimeError(str(value) + ' is not a number')
    return value

#This changes unicode characters into the normal ones
def restoreUnicode(textString):
    textString = textString.replace("&#x201c;," '"')
    textString = textString.replace("&#x201d;," '"')
    textString = textString.replace("&quot;," '"')
    textString = textString.replace("&#xfffd;," 'degrees')
    textString = textString.replace("&#x2019;," "'")
    textString = textString.replace('&lt;', '<')
    textString = textString.replace('&gt;', '>')
    return textString

#Check is stringFramentB comes after stringFragmentA
def checkBAfterA(stringIn, stringFragmentB, stringFragmentA):
    #Need to check placement
    if stringFragmentB == stringFragmentA:
        tempArray = stringIn.split(stringFragmentB)

        if len(tempArray) > 2:
```

```
            return True
        else:
            return False

    #Verify fragments are in the string
    if stringFragmentB in stringIn and stringFragmentA in stringIn:
        indexForB = stringIn.index(stringFragmentB)
        indexForA = stringIn.index(stringFragmentA)

        if indexForA < indexForB:
            return True
        else:
            return False
    #One of the values for checkBAfterA function was not in the
    #  input string. False was returned by default.
    else:
        return False

#Returns the first string between to characters
def extractTextBetween(stringIn, startText, endText):
    if startText in stringIn:
            leftSpot = stringIn.index(startText)

            if endText not in stringIn:
                return stringIn[leftSpot + len(startText):]

            for i in range(len(stringIn)):
                if (leftSpot + len(startText) + i) < len(stringIn) and \
                        stringIn[(leftSpot + len(startText) + i): (leftSpot + \
                        len(startText) + i + len(endText))] == endText:

                    newEntry =
                                stringIn[leftSpot+len(startText):leftSpot+len(startText)+i]
                    return newEntry
                elif (leftSpot + len(startText) + i) == len(stringIn):
                    return stringIn[leftSpot + len(startText):]
    else:
        return stringIn

# Verifies if given text has a Number followed by a .”“
def hasStepNumber(textIn):
    textIn = textIn.strip()

    if .”“ not in textIn and “(“ != textIn[0]:
        return False
    #Prevents e.g., from being recognized as step number
    elif “e.g.” in textIn[:6] or “e. g.” in textIn[:6]:
        return False
    #This checks for steps that are enclosed by parenthesis
    elif “(“ == textIn[0] and “)” in textIn:
        checkText = extractTextBetween(textIn, “(,” “)”).strip()

        if checkText.isnumeric():
            return True
        else:
            return False
    elif .”“ not in textIn:
        return False

    partsArray = textIn.split(.”“)

    #Return false if an item were of the form Number.Number, e.g. 1.1
    if partsArray[1].isnumeric():
        return False

    #Verify that punctuation and special characters are not in the
    # prior to the period.
    for i in range(len(PUNCTUATION)):
        if PUNCTUATION[i] in partsArray[0]:
```

```
            return False
    for i in range(len(OTHER_SPECIAL_CHARACTERS)):
        if OTHER_SPECIAL_CHARACTERS[i] in partsArray[0]:
            return False

    # This assumes letter steps only have a single character
    if len(partsArray[0]) < 2 and checkForCapitalFirstCharacter(partsArray[1]):
        return True
    # Return true if the the step number is a normal number
    # Assume that a process will have less than 50 steps
    elif partsArray[0].isnumeric():
        if int(partsArray[0]) < 50 and \
                            checkForCapitalFirstCharacter(partsArray[1]):
            return True
        else:
            return False

    # This assumes that if a step number is 2 or longer, then it must
    #  be a Roman Numeral.
    # If there is a number mixed in, then it is not a Roman Numeral
    else:
        for i in range(len(NOT_ROMAN_NUMBERS)):
            if NOT_ROMAN_NUMBERS[i] in partsArray[0].upper():
                return False

        for i in range(len(NUMBERS)):
            if str(NUMBERS[i]) in partsArray[0].upper():
                return False
    if checkForCapitalFirstCharacter(partsArray[1]):
        return True
    else:
        return False

# Check to see if the given text is a note
def isNote(textIn):
    if ":" not in textIn:
        return False

    partsArray = textIn.split(":")

    # Notes will have all CAPS for the first portion
    if (isAllCaps(partsArray[0])):
        return True
    # If there are not all CAPS, then it should not be a Note field
    else:
        return False

# Function checks if the textIn is all upper cased and returns a boolean
# True if it is all upper cased, False otherwise
def isAllCaps(textIn):
    if textIn == textIn.upper():
        return True
    else:
        return False

#Return a string without the step number
def removeStepNumber(textIn):
    #Verify that the text has a step number before continuing
    if hasStepNumber(textIn):
        if textIn[0] == "(":
            for i in range(len(textIn)):
                if textIn[i] == ")":
                    return (textIn[i+1:]).strip()
        else:
            firstPeriodLocation = (textIn.index("."))

            #Return the string without the step number and removes extra spaces
            return (textIn[firstPeriodLocation+1:]).strip()
    #If there is not step number, just return the original text
```

```
    else:
        return textIn

#Replace last punctuation mark with a question mark
def makeIntoQuestions(textIn):
    stringLength = len(textIn)

    #Looks at the last spot in text for punctuation
    if textIn[stringLength -1] in SHORT_PUNCTUATION:
        return textIn[: stringLength - 1] + QUESTION_MARK_STRING
    #Looks at the second to last spot for punctuation
    elif textIn[stringLength -2] in SHORT_PUNCTUATION:
        return textIn[: stringLength - 2] + QUESTION_MARK_STRING + \
                                                textIn[stringLength -1]
    #Catch all case to append a question mark to the end of text
    else:
        return textIn + QUESTION_MARK_STRING

def checkForCapitalFirstCharacter(textIn):
    strippedText = textIn.strip()
    indexLetter = 0

    if strippedText == "":
        return False
    elif strippedText[0] == '"':
        indexLetter = 1

    if strippedText[indexLetter] == strippedText[indexLetter].upper() and\
                                strippedText[indexLetter] not in PUNCTUATION:
        return True
    else:
        return False
```

LIST OF REFERENCES

Adelberg, B. (1998). NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. *ACM SIGMOD Record 27*, 2 (June 1998), 283–294. https://doi.org/10.1145/276305.276330

Artifex Software (n.d.). Mutool convert. Retrieved January 22, 2018 from https://mupdf.com/docs/manual-mutool-convert.html

Azamov, N. (2015). MATLAB based language for generating randomized multiple choice questions. Retrieved from Cornell University Library: arXiv:1502.02348

Bjordahl, M., Talboy, R., Neyman, J., McLaughlin, T., & Hoenike, R. (2014). Effect of a direct instruction flashcard system for increasing the performance of basic division facts for a middle school student with ADD/OHI. *I-Manager's Journal on Educational Psychology, 8*(2), 7–14.

Bork, A. (1980). Preparing on-line quizzes. ACM SIGCUE Outlook, 14(4), 2–16. doi:10.1145/964076.964077

Bruce-Lockhart, M., Norvell, T., & Crescenzi, P. (2009). Adding test generation to the teaching machine. *ACM Transactions on Computing Education (TOCE), 9(2), 1–14*. https://doi.org/10.1145/1538234.1538239

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to algorithms, third edition.* Cambridge, Massachusetts: The MIT Press.

Country Living (2017, February 15). Best vanilla cake. Retrieved May 30, 2018 from https://www.countryliving.com/food/drinks/recipes/a871/basic vanilla cake 69.

Dong, C., & Liu, X. (2013). A mobile app for learning Japanese. Knowledge sharing through technology. *8th International Conference on Information and Communication Technology in Teaching and Learning, ICT 2013, Hong Kong, China, July 10 - 11, 2013. Revised selected papers.* (Vol. 158–166, pp. 158–166). https://doi.org/10.1007/978-3-642-45272-7_15

Fang, Y., Bo, L., & Ge, Y. (2006). A study on information extraction from PDF files. *Advances In Machine Learning And Cybernetics*, 3930, 258–267. Retrieved from https://link.springer.com/chapter/10.1007/11739685_27

Hardeniya, N. (2015). *NLTK essentials* [Safari Online Version]. Retrieved from http://techbus.safaribooksonline.com/

Hürst, W., Jung, S., & Welte, M. (2007). Effective learn-quiz generation for handheld devices. Proceedings of the 9th international conference on human computer interaction with mobile devices and services (pp. 364–366). ACM. doi:10.1145/1377999.1378036

Kandar, Shyamalendu. (2013). *Introduction to Automata Theory, Formal Languages and Computation.* Retrieved from http://techbus.safaribooksonline.com/book/information-technology-and-software-development/9788131793510

Kromkamp, B. (2018). Knowledge management, Python tree implementation. Retrieved October 17, 2017 from http://www.quesucede.com/page/show/id/python-3-tree-implementation

Kupzyk, S., Daly,Edward J., I.,II, & Andersen, M. N. (2011). A comparison of two flash card methods for improving sight-word reading. *Journal of Applied Behavior Analysis*, *44*(4), 781–92. Retrieved from http://libproxy.nps.edu/login?url=https://search.proquest.com.libproxy.nps.edu/docview/918647788?accountid=12702

MacQuarrie, L. L., Tucker, J. A., Burns, M. K., & Hartman, B. (2002). Comparison of retention rates using traditional, drill sandwich, and incremental rehearsal, flash card methods. *School Psychology Review*, *31*(4), 584. Retrieved from http://libproxy.nps.edu/login?url=https://search.proquest.com.libproxy.nps.edu/docview/219653513?accountid=12702

Mccord, M., Murdock, J., & Boguraev, B. (2012). Deep parsing in Watson. *IBM Journal Of Research And Development*, *56*(3-4). https://doi.org/10.1147/JRD.2012.2185409

Naval Sea Systems Command. (n.d.). Engineering Operational Sequencing System DTD. Retrieved October 16, 2017 from http://www.navsea.navy.mil/Home/Warfare-Centers/NSWC-Carderock/Resources/Technical-Information-Systems/Navy-XML-SGML-Repository/DTDs-Schemas/EOSS/Poruban, J., Forgac, M., Sabo, M., & Behalek, M. (2010). Annotation Based Parser Generator. Computer Science And Information Systems, 7(2). https://doi.org/10.2298/CSIS1002291P

Ramakrishnan Cartic, Patnia Abhishek, Hovy Eduard, & Burns Gully APC. (2012). Layout-aware text extraction from full-text PDF of scientific articles. Source Code for Biology and Medicine, 7(1), 7. https://doi.org/10.1186/1751-0473-7-7. Retrieved from https://scfbm.biomedcentral.com/track/pdf/10.1186/1751-0473-7-7

Sakamoto, N. (2017). Automated Generation of Fill-in-the-Blanks-Type Quizzes Using Wikipedia. International Journal of Computer Theory and Engineering, 9(5), 367–373. https://doi.org/10.7763/IJCTE.2017.V9.1168. Retrieved from http://www.ijcte.org/vol9/1168-T052.pdf

Shipman, J. (2013, December 31). Tkinter 8.5 reference: a GUI for Python, 2. a minimal application. Retrieved October 25, 2017 from http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/minimal-app.html

Thanaki, J. (2017). *Python Natural Language Processing*. [Safari Online Version] Retrieved from http://techbus.safaribooksonline.com/

Towns, M. (2014). Guide to developing high quality, reliable, and valid multiple-choice assessments. *Journal of Chemical Education*, 91(9), 1426–1431. Retrieved from https://pubs.acs.org.libproxy.nps.edu/toc/jceda8/91/9

Turenne, N. (2013). *Knowledge Needs and Information Extraction: Towards an Artificial Consciousness* [Safari Online Version]. Retrieved from http://techbus.safaribooksonline.com/

Tutorials Point (2018, April 21). Python - GUI Programming (Tkinter). Retrieved from https://www.tutorialspoint.com/python/python_gui_programming.htm

W3Schools (n.d.). HTML element reference. Retrieved April 15, 2018 from https://www.w3schools.com/TAGs/ref_byfunc.asp

Welbl, J., Liu, N., & Gardner, M. (2017). Crowdsourcing Multiple Choice Science Questions. Retrieved from Cornell University Library: arXiv:1707.06209v1

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California