Practice Exercise 1: Write a program to read through a file and print the contents of the file (line by line) all in upper case. Executing the program will look as follows: (mbox-data).

```python
#Practice Exercise 1
def print_file_contents_uppercase(file_name):
    try:
        with open(file_name, 'r') as file:
            for line in file:
                print(line.upper().rstrip())
    except FileNotFoundError:
        print(f"Error: File '{file_name}' not found.")
    except IOError:
        print(f"Error: Unable to read '{file_name}'.")

if __name__ == "__main__":
    file_name =input("Enter your file name:")
    print_file_contents_uppercase(file_name)
```

```
Enter your file name:mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN  5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
         BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
         SAT, 05 JAN 2008 09:14:16 -0500
X-SIEVE: CMU SIEVE 2.3
RECEIVED: FROM MURDER ([UNIX SOCKET])
         BY MAIL.UMICH.EDU (CYRUS V2.2.12) WITH LMTPA;
         SAT, 05 JAN 2008 09:14:16 -0500
RECEIVED: FROM HOLES.MR.ITD.UMICH.EDU (HOLES.MR.ITD.UMICH.EDU [141.211.14.79])
        BY FLAWLESS.MAIL.UMICH.EDU () WITH ESMTP ID M05EEFR1013674;
        SAT, 5 JAN 2008 09:14:15 -0500
RECEIVED: FROM PAPLOO.UHI.AC.UK (APP1.PROD.COLLAB.UHI.AC.UK [194.35.219.184])
        BY HOLES.MR.ITD.UMICH.EDU ID 477F90B0.2DB2F.12494 ;
         5 JAN 2008 09:14:10 -0500
```

Figure 01: program 1

Practice Exercise 2: Write a program to prompt for a file name, and then read through the file and look for lines of the form: X-DSPAM-Confidence: 0.8475 When you encounter a line that starts with “X-DSPAM-Confidence:” pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence. (mbox data).

```
#Practice Exercise 2

def extract_spam_confidence(line):
    # Extract the floating-point number from the line
    confidence_str = line[line.find(':') + 1:].strip()
    try:
        return float(confidence_str)
    except ValueError:
        return None
def main():
    file_name = input("Enter the file name: ")
    try:
        with open(file_name, 'r') as file:
            total_confidence = 0.0
            count = 0
            for line in file:
                if line.startswith('X-DSPAM-Confidence:'):
                    confidence = extract_spam_confidence(line)
                    if confidence is not None:
                        total_confidence += confidence
                        count += 1
            if count > 0:
                average_confidence = total_confidence / count
                print("Average spam confidence:", average_confidence)
            else:
                print("No X-DSPAM-Confidence lines found in the file.")
    except FileNotFoundError:
        print("File not found. Please check the file name and try again.")
    except Exception as e:
        print("An error occurred:", str(e))
if __name__ == "__main__":
    main()
```

```
Enter the file name: mbox-short.txt
Average spam confidence: 0.7507185185185187
```

Figure 02: program 2

Practice Exercise 3: Write a program to open the file romeo.txt and read it line by line. For each line, split the line into a list of words using the split function. For each word, check to see if the word is already in the list of unique words. If the word is not in the list of unique words, add it to the list. When the program completes, sort and print the list of unique words in alphabetical order.

```
# Practice Exercise 3
def main():
    file_name = input("Enter file: ")
    unique_words = []
    try:
        with open(file_name, 'r') as file:
            for line in file:
                words = line.split()
                for word in words:
                    word = word.strip('.,!?;:()"')
                    if word.lower() not in unique_words:
                        unique_words.append(word.lower())
        unique_words.sort()
        print_unique_words(unique_words)
    except FileNotFoundError:
        print("File not found. Please make sure the file exists in the specified location.")
    except Exception as e:
        print(f"An error occurred: {e}")
def print_unique_words(unique_words):
    for word in unique_words:
        print(f"'{word}',", end=' ')
if __name__ == "__main__":
    main()
```

```
Enter file: romeo.txt
'already', 'and', 'arise', 'breaks', 'but', 'east', 'envious', 'fair', 'grief', 'is', 'it', 'juliet', 'kill', 'light', 'moon',
```

Figure 03: program 3

Practice Exercise 4: Remove the following stopwords from the romeo.txt file and then find the list of unique words. stp = [”But”,’is’,’the’,’with’,’and’]

```
# Practice Exercise 4
def main():
    file_name = input("Enter file: ")
    stp = ["But", "is", "the", "with", "and"]
    unique_words = []
    try:
        with open(file_name, 'r') as file:
            for line in file:
                words = line.split()
                for word in words:
                    word = word.strip('.,!?;:()"')
                    if word.lower() not in stp and word.lower() not in unique_words:
                        unique_words.append(word.lower())
        unique_words.sort()
        print_unique_words(unique_words)
    except FileNotFoundError:
        print("File not found")
    except Exception as e:
        print(f"An error occurred: {e}")
def print_unique_words(unique_words):
    for word in unique_words:
        print(f"'{word}',", end=' ')
if __name__ == "__main__":
    main()
```

```
Enter file: romeo.txt
'already', 'arise', 'breaks', 'but', 'east', 'envious', 'fair', 'grief', 'it', 'juliet', 'kill', 'light', 'moon', 'pale', 'sick', '
```

Figure 04: program 4

Practice Exercise 5: Write a program to read through the mailbox data and when you find the line that starts with “From”, you will split the line into words using the split function. We are interested in who sent the message, which is the second word on the From line. (mbox-data).

```
# Practice Excercise 5
def read_mbox_file(filename):
    sender_emails = []
    from_lines_count = 0
    try:
        with open(filename, 'r') as mbox_file:
            for line in mbox_file:
                if line.startswith("From "):
                    from_lines_count += 1
                    words = line.split()
                    if len(words) >= 2:
                        sender_emails.append(words[1])
    except FileNotFoundError:
        print("Error: File not found.")
        return
    for email in sender_emails:
        print(email)
    print("There were", from_lines_count, "lines in the file with From as the first word.")
if __name__ == "__main__":
    file_name = input("Enter a file name: ")
    read_mbox_file(file_name)
```

```
Enter a file name: mbox-short.txt
stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu
rjlowe@iupui.edu
zqian@umich.edu
rjlowe@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
gsilver@umich.edu
gsilver@umich.edu
zqian@umich.edu
```

Figure 05: program 5

Practice Exercise 6: Rewrite the program that prompts the user for a list of numbers and prints out the maximum and minimum of the numbers at the end when the user enters “done”. Write the program to store the numbers the user enters in a list and use the max() and min() functions to compute the maximum and minimum numbers after the loop completes.

```python
# Practice Excercise 6
def find_max_min(numbers):
    if numbers:
        max_num = max(numbers)
        min_num = min(numbers)
        return max_num, min_num
    else:
        return None, None
if __name__ == "__main__":
    numbers_list = []
    while True:
        user_input = input("Enter a number: ")
        if user_input.lower() == 'done':
            break
        try:
            number = float(user_input)
            numbers_list.append(number)
        except ValueError:
            print("Invalid input. Please enter a valid number or 'done' to finish.")
    max_number, min_number = find_max_min(numbers_list)
    if max_number is not None and min_number is not None:
        print("Maximum:", max_number)
        print("Minimum:", min_number)
    else:
        print("No numbers were entered.")
```

```
Enter a number: 6
Enter a number: 2
Enter a number: 9
Enter a number: 3
Enter a number: 5
Enter a number: done
Maximum: 9.0
Minimum: 2.0
```

Figure 06: program 6