```python
import numpy as np
import pandas as pd

### Problem 1a ###

# Read lines from file and save as pandas dataframe. Separate lines into two columns.
data = pd.read_csv('hw1_word_counts_05.txt', sep=" ", header=None, names=['Word', 'Count'])

# Compute prior probability and add as third column of dataframe for each word
data['P(W=w)'] = data['Count']*1.0/data['Count'].sum()

# Sort the dataframe by the word's prior probability (Second column of dataframe)
data_sorted = data.sort_values(by = ['P(W=w)'], ascending=False)

# Sanity check. 15 most frequent and 14 least frequent words.
print(data_sorted.head(15))
print(data_sorted.tail(14))

### Problem 1b ###

# Make words and prior probabilities from pandas dataframe to lists
words = data['Word'].tolist()
priors = data['P(W=w)'].to_numpy()
alphabet = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']

# Check if word is possible given the evidence
def isWordPossible(word, correct_evidence, incorrect_evidence):
    for i, letter in enumerate(word):
        if correct_evidence[i] == None:
            if letter in correct_evidence:
                return 0
        if correct_evidence[i] != None:
            if letter != correct_evidence[i]:
                return 0
        if letter in incorrect_evidence:
            return 0
    return 1

# Evaluate the denominator in posterior probability
def posteriorDenominator(words, priors, correct_evidence, incorrect_evidence):
    denominator = 0.0
    for i, word in enumerate(words):
        denominator += isWordPossible(word,correct_evidence,incorrect_evidence)*priors[i]
    return denominator

# Evaluate posterior probability for all words
def posteriorProbability(words, priors, correct_evidence,
```

```
def posteriorProbability(words, priors, correct_evidence,
incorrect_evidence):
    posteriors = []
    denominator = posteriorDenominator(words, priors, correct_evidence,
incorrect_evidence)
    for i, word in enumerate(words):
        nominator =
isWordPossible(word,correct_evidence,incorrect_evidence)*priors[i]
        posteriors.append(nominator*1.0/denominator*1.0)
    return posteriors

# Check if a letter is in a word
def isLetterInWord(letter, word):
    for char in word:
        if letter == char:
            return 1
    return 0

# Predict the probability of a single letter
def letterPredictiveProbability(posteriors, letter, words, priors,
correct_evidence, incorrect_evidence):
    posteriors = posteriorProbability(words, priors, correct_evidence,
incorrect_evidence)
    letter_prob = 0.0
    for i, word in enumerate(words):
        letter_in_word = isLetterInWord(letter,word)
        letter_prob += letter_in_word*posteriors[i]
    if letter in correct_evidence or letter in incorrect_evidence:
        letter_prob = 0
    return letter_prob

# Predict probability of all letters, returning the letter with largest
probability
def nextGuess(alphabet, words, priors, correct_evidence, incorrect_evidence):
    print("Correct evidence: ", correct_evidence)
    print("Incorrect evidence: ", incorrect_evidence)
    max_probability = 0.0
    max_letter = ''
    posteriors =
posteriorProbability(words,priors,correct_evidence,incorrect_evidence)
    for i, alpha in enumerate(alphabet):
        letter_probability =
letterPredictiveProbability(posteriors,alpha,words,priors,correct_evidence,in
correct_evidence)
        if letter_probability > max_probability:
            max_probability = letter_probability
            max_letter = alphabet[i]
    print("Next letter: ", max_letter, " with probability: ",
max_probability, '\n')

# TEST CASES
print("Test case 1")
correct_evidence = [None,None,None,None,None]
incorrect_evidence = []
nextGuess(alphabet,words,priors,correct_evidence,incorrect_evidence)
```

```
print("Test case 2")
correct_evidence = [None,None,None,None,None]
incorrect_evidence = ['E','A']
nextGuess(alphabet,words,priors,correct_evidence,incorrect_evidence)

print("Test case 3")
correct_evidence = ['A',None,None,None,'S']
incorrect_evidence = []
nextGuess(alphabet,words,priors,correct_evidence,incorrect_evidence)

print("Test case 4")
correct_evidence = ['A',None,None,None,'S']
incorrect_evidence = ['I']
nextGuess(alphabet,words,priors,correct_evidence,incorrect_evidence)

print("Test case 5")
correct_evidence = [None,None,'O',None,None]
incorrect_evidence = ['A','E','M','N','T']
nextGuess(alphabet,words,priors,correct_evidence,incorrect_evidence)
```