Prototype Application Code

```
import string, re

class NameProbabilityPair:
    #used in computing probabilities using Bayes theorem
    def __init__(self, name, value):
        self.name = name
        self.value = value
    def compare(self, comp):
        return self if self.value > comp.value else comp

def clean_text(input):
    #removes punctuations
    input = re.sub(r'[^\w\s]','',input)
    lines = input.lower().split("\n")

    return lines

def create_bag(input):
    bag = {}

    #filling word bag
    for line in input:
        text = line.split(" ")
        for word in text:
            if word not in bag:
                bag[word] = 0
            #increment occurence in word bag
            bag[word] = bag[word]+1

    return bag

def print_bag(bag):
    index = 0
    for word in bag:
        print(str(index)+"\t\t"+word+"\t\t"+str(bag[word]))
        index = index+1

def convert_to_chain(clean_text):
    #splits the entire text into a tokenized form.
    word_list = []
    for line in clean_text:
        text = line.split(" ")
        word_list = word_list + text
    return word_list

def occurence_count(matrix, chain):
    #counts word appearance given another word
    for i in range(1, len(chain)):
        matrix[chain[i]][chain[i-1]] = matrix[chain[i]][chain[i-1]] + 1

def divide_by_occurences(matrix, bag):
    #converts the occurence counts into a probability
    for key in matrix.keys():
        for key2 in matrix[key].keys():
            matrix[key][key2] = matrix[key][key2]/bag[key]
    return matrix

def gen_prob_matrix(bag, chain):
    #creates the matrix with a size of X by X, where X is the amount of words in the word bag
    matrix = {}
    for key in bag.keys():
        matrix[key] = {}
        for key2 in bag.keys():
            matrix[key][key2] = 0

    occurence_count(matrix, chain)

    return matrix

def print_prob_matrix(matrix):
    for key in matrix.keys():
        for key2 in matrix[key].keys():
            print(key, "given", key2, "=", matrix[key][key2])

def get_top_rankers(word, slots, matrix, chain, bag):
    #creates name-probability pairs and uses those to rank the probabilities
    ranking = []
    p_word = bag[word]/len(text_chain)
    for key2 in matrix[word].keys():
        p_key2 = bag[key2]/len(text_chain)
        p_word_key2 = matrix[key2][word]
        converted_pair = NameProbabilityPair(key2, ((p_word_key2*p_key2)/p_word))
        ranking = ranking + [converted_pair]
    ranking = sorted(ranking, key=lambda pair: pair.value)
    return ranking[::-1]

############### main function ###############

original = open("hp1.txt").read();
punc_free = clean_text(original)
split_bag = create_bag(punc_free)

text_chain = convert_to_chain(punc_free)

probabilities = gen_prob_matrix(split_bag, text_chain)
probabilities = divide_by_occurences(probabilities, split_bag)

word = "harry"
rankings = 10

rankers = get_top_rankers(word, rankings, probabilities, text_chain, split_bag)

print("Top", rankings, "Possible Words for", word + ":")
for i in range(0, rankings):
    print(rankers[i].name, ":", rankers[i].value)
```