```python
# HW2
# Name: William Martinez
# Collaborators: None
# Date:4/21/24

import random

### Problem 1: Character Frequency in a string
def count_characters(s):
    """
    Return the frequency of each character in a string.
    Args:
        s: A string.
    Return:
        char_count: A dictionary of all characters in a string and their
        frequencies.
    """
    ## Initialization of Variables:
    char_count = dict()
    ## Character Frequency in a String:
    # If key not in dict, append it with a default value of 1, and if the key
    # already exist, increment the dict value by 1 for each occurrence.
    for i in range(len(s)):
        char = s[i]
        char_count[char] = char_count.get(char, 0) + 1
    return(char_count)

### Problem 2: N-Gram Frequency in a string
def count_ngrams(s, n = 1):
    """
    Return the frequency of each unique n-gram in a string.
    Args:
        s: A string.
        n: An integer. Corresponds to the length of an n-gram.
    Return:
        ngram_count: A dictionary of all characters in a string and their
        frequencies.
    """
    ## Initialization of Variables:
    ngram_count = dict()
    ## Character Frequency in a String:
    # Subtract the range of the for-loop by the value of n and add 1.
    # If ngram is not in dict, append it with a default value of 1. If the ngram
    # is in dict, increment the dict value by 1 for each occurrence.
    for i in range(len(s) - n + 1):
        ngram_str = s[i: i + n]
        ngram_count[ngram_str] = ngram_count.get(ngram_str, 0) + 1
    return(ngram_count)

### Problem 3: Generative Text Using the Markov Text Method
def markov_text(s, n, length = 100, seed = "Emma Woodhouse"):
```

```python
"""
Return synthetically generated text using the Markov generative text method.
The relative frequency of (n+1)-grams that match the last n-gram of the seed
are used as weights in a random choice for the next character to be appended
to the seed.
Args:
    s: A string. This string will be used to create a frequency dictionary
    of (n+1)grams.
    n: An integer. The ngram length.
    length: An integer. The character length of the generative text.
    seed: A string. A subset of the string, s. The start of the generative
    text.
Return:
    gen_text: A string. Starts with seed and each character after is
    generated using the Markov method.
"""
## Error Catch:
# When n is greater than the seed length.
if (len(seed) < n):
    raise ValueError("Length of seed is less than size of n")
all_ngrams = count_ngrams(s,n+1) # A list of all ngrams that are .
gen_text = seed # start the generated text with seed.
## Generated Text Loop:
# Loop until gen_text is equal or greater than length.
while len(gen_text) < length:
    ## Initialization of Variables in Loop.
    # Theses variables will re-initialize with each loop
    last_ngram = gen_text[-n:] # Last ngram from gen_text
    options = list()
    weights = list()
    total = int()
    ## Match the begining n characters of the (n+1)gram with the characters
    ## of last ngram of the last generated text:
    for key, value in all_ngrams.items():
        if key[:-1] == last_ngram:
            options.append(key)
            weights.append(value)
            total += value
        else:
            continue
    # Sum of matched (n+1)grams frequencies.
    total = sum(weights)
    # Calculate relative frequencies (weights).
    for i in range(len(weights)):
        weights[i] = weights[i] / total
    # Randomly select a matched (n+1)gram based on its weight. keep the last
    # character.
    gen_ngram = random.choices(options, weights)[0][-1]
    # append the last character to the generated text
    gen_text += gen_ngram # append to generated text
return(gen_text)
```