MTH 3300 Problem Set 5

Due date: Mar 17, 2025 at 11:59PM

For the programming assignment, please follow the following naming convention for your Python files.
`mth3300_<lastname>_<firstname>_pset5_part<#>.py`

As with previous problem sets, the written portion should be a separate PDF.

Note that for this homework, I expect that all will provide fully-implemented functions that have at least passed the basic tests I've provided.

1. (15 points) Write a function `find_factors` that takes an integer `n` and returns a **unique list** of all its factors. Note you will get points taken off if your return value does not have a type of `list`. Your resulting list should be sorted in ascending order.

```python
def find_factors(n):
    factors = []
    for i in range(1, n + 1):
        if n % i == 0:
            factors.append(i)
    return factors

assert find_factors(8) == [1, 2, 4, 8]
assert find_factors(10) == [1, 2, 5, 10]
assert find_factors(7) == [1, 7]
assert find_factors(1) == [1]
```

2. (20 points) Write a function `k_most_frequent` that takes a list called `items` and another argument `k`. Your objective is to implement a function that keeps track of the frequency an item appears in the list and then return the `k` most frequent elements in descending order. Note that you are not allowed to use the `collections.Counter` data structure for this problem. If there is a tie between two values in terms of frequency, then the smaller value should be considered to have priority.

```python
def k_most_frequent(items, k):
    freq = {}
    for item in items:
        freq[item] = freq.get(item, 0) + 1
    sorted_items = sorted(freq.items(), key=lambda x: (-x[1], x[0]))
    return [item for item, _ in sorted_items[:k]]

assert k_most_frequent([1, 1, 1, 2, 2, 3], 2) == [1, 2]
assert k_most_frequent([1, 1, 2, 2, 3, 3], 2) == [1, 2]
assert k_most_frequent([5, 3, 1, 1, 5, 5, 3], 3) == [5, 1, 3]
assert k_most_frequent([4, 4, 2, 2, 2], 3) == [2, 4]
assert k_most_frequent([7, 7, 7, 7], 1) == [7]
assert k_most_frequent([], 3) == []
```

3. (20 points) Write a function `group_by_first_letter` that takes a list of strings called `words`. Your objective is to implement a function that groups each word by its first character. In other words, return a dictionary where the key is the first character of a word and the value is a list of words that all start with the same character. Another requirement here is to make sure that the list of words are **sorted** in alphabetical order.

```python
def group_by_first_letter(words):
    result = {}
    for word in words:
        first_letter = word[0]
        if first_letter not in result:
            result[first_letter] = []
        result[first_letter].append(word)
    for key in result:
        result[key].sort()
  return result

assert group_by_first_letter(["abc", "ad", "b", "cde", "ca"]) == {
    "a": ["abc", "ad"],
    "b": ["b"],
    "c": ["ca", "cde"]
}
```

```
assert group_by_first_letter(["123", "456", "apple", "1banana"]) == {
    "1": ["123", "1banana"],
    "4": ["456"],
    "a": ["apple"]
}
```

4. Linguistic researchers from the North Pole University have discovered a collection of ancient scrolls in a hidden ice cave. They believe these documents might contain clues about the origins of elfin languages, but they need your help to analyze them.

   Each scroll contains text that has been damaged, with only certain characters remaining legible. The researchers need to create a "fingerprint" of each document based on the frequency and distribution of these remaining characters.

   (a) (5 points) Write a function `get_character_frequency_from_scroll` that takes one argument `scroll` and returns the frequency of the characters within the scroll

```
def get_character_frequency_from_scroll(scroll):
    frequencies = {}
    for char in scroll:
        if char not in frequencies:
            frequencies[char] = 0
        frequencies[char] += 1
    return frequencies

assert get_character_frequency_from_scroll("abcabcdefgahijk") == {
    'a': 3, 'b': 2, 'c': 2, 'd': 1, 'e': 1, 'f': 1, 'g': 1, 'h': 1, 'i': 1, 'j': 1, 'k': 1
}
```

   (b) (5 points) Write a function `get_signature_set` that takes the frequencies you derived from part (a). The "signature set" of each scroll is a set of characters appears exactly 3 times within the scroll.

```
def get_signature_set(frequencies):
    signature_set = set()
    for char, freq in frequencies.items():
        if freq == 3:
            signature_set.add(char)
    return signature_set

assert get_signature_set({
    'a': 3, 'b': 2, 'c': 2, 'd': 1, 'e': 1, 'f': 1, 'g': 1, 'h': 1, 'i': 1, 'j': 1, 'k': 1
}) == {'a'}
```

   (c) (5 points) Each scroll has a document value which is the product between the number of unique characters in the scroll and the size of signature set. Write a function `get_document_value` using also the functions you've created from parts (a) and (b) to derive the document value.

```
def get_document_value(scroll):
    frequencies = get_character_frequency_from_scroll(scroll)
    signature_set = get_signature_set(frequencies)
    return len(frequencies) * len(signature_set)
```