Daily Coding Problem #28

Problem

This problem was asked by Palantir.

Write an algorithm to justify text. Given a sequence of words and an integer line length k, return a list of strings which represents each line, fully justified.

More specifically, you should have as many words as possible in each line. There should be at least one space between each word. Pad extra spaces when necessary so that each line has exactly length k. Spaces should be distributed as equally as possible, with the extra spaces, if any, distributed starting from the left.

If you can only fit one word on a line, then you should pad the right-hand side with spaces.

Each word is guaranteed not to be longer than k.

For example, given the list of words ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"] and k = 16, you should return the following:

```
["the  quick brown", # 1 extra space on the left
"fox  jumps  over", # 2 extra spaces distributed evenly
"the   lazy   dog"] # 4 extra spaces distributed evenly
```

Solution

It seems like the justification algorithm is independent from the groupings, so immediately we should figure out two things:

- How to group lines together so that it is as close to k as possible (without going over)
- Given a grouping of lines, justifying the text by appropriately distributing spaces

To solve the first part, let's write a function `group_lines` that takes in all the words in our input sequence as well as out target line length k, and return a list of list of words that represents the lines that we will eventually justify. Our main strategy will be to iterate over all the words, keep a list of words for the current line, and because we want to fit as many words as possible per line, estimate the current line length, assuming only one space between each word. Once we go over k, then save the word and start a new line with it. So our function will look something like this:

```
def min_line(words):
    return ' '.join(words)

def group_lines(words, k):
    '''
    Returns groupings of |words| whose total length, including 1 space in between,
    is less than |k|.
    '''
    groups = []
    current_sum = 0
    current_line = []
    for i, word in enumerate(wordwordss):
        # Check if adding the next word would push it over
        # the limit. If it does, then add |current_line| to
        # group. Also reset |current_line| properly.
        if len(min_line(current_line + [word])) > k:
            groups.append(current_line)
            current_line = []
        current_line.append(word)

    # Add the last line to groups.
    groups.append(current_line)
    return groups
```

Then, we'll want to actually justify each line. We know for sure each line we feed from `group_lines` is the maximum number of words we can pack into a line and no more. What we can do is first figure out how many spaces we have available to distribute between each word. Then from that, we can calculate how much base space we should have between each word by dividing it by the number of words minus one. If there are any leftover spaces to distribute, then we can keep track of that in a counter, and as we rope in each new word we'll add the appropriate number of spaces. We can't add more than one leftover space per word.

```
def justify(words, length):
    '''
    Precondition: |words| can fit in |length|.
    Justifies the words using the following algorithm:
        - Find the smallest spacing between each word (available_spaces / spaces)
        - Add a leftover space one-by-one until we run out
    '''
    if len(words) == 1:
        word = words[0]
        num_spaces = length - len(word)
        spaces = ' ' * num_spaces
        return word + spaces
    spaces_to_distribute = length - sum(len(word) for word in words)
    number_of_spaces = len(words) - 1
    smallest_space = floor(spaces_to_distribute / number_of_spaces)
    leftover_spaces = spaces_to_distribute - (number_of_spaces * smallest_space)
    justified_words = []
    for word in words:
        justified_words.append(word)
        current_space = ' ' * smallest_space
        if leftover_spaces > 0:
            current_space += ' '
            leftover_spaces -= 1
        justified_words.append(current_space)
    return ''.join(justified_words).rstrip()
```

The final solution should just combine our two functions:

```
def justify_text(words, k):
    return [justify(group, k) for group in group_lines(words, k)]
```