File: Design.md
Design for all the sentence methods
Created: September 2024
Developer: Swathi Danturi
Date: 9/23/2024

Design of `my_split()` core function

- define `my_split()` method and `self` is the current instance of the class `Sentence`
- `self.sentence` is an instance variable of the class `Sentence` and is initialized to a string in the constructor, which is passed as an argument when the class instance is created
- use the instance variable `self.words` as an accumulator to store the list of words from string split
- declare `a_word` an accumulator to store each word from `self.sentence` and initialize it to an empty string
- `for` each character `char` in `self.sentence`, iterate through the following statements:
    - check if `char` is space `' '`
    - if `char` is not a space then:
        - add `char` to the `a_word`, an accumulator of type string to store the characters
    - else:
        - if `a_word` is not empty then:
            - append `a_word` to `self.words`
            - empty `a_word` by assigning an empty string to it
- if `a_word` is not empty then append it to `self.words`

Design of `my_join()` core function

- define `my_join()` method and `self` is the current instance of the class `Sentence`
- `self.sentence` is an instance variable of the class `Sentence` and is initialized to a string in the constructor, which is passed as an argument when the class instance is created
- call the method `my_split()` to split `self.sentence` and the result list will be stored in `self.words` (this is the functionality of `my_split()`)
- declare an accumulator `joined_string` and initialize it to an empty string to store the concatenated string of the word list `self.words`
- for each `word` in `self.words`, iterate through the following:
    - concatenate `word` to `joined_string` using `+` operator
    - check if `word` is the last word in the list using reverse indexing:
        - if it is not the last word, then add a space `' '` to `joined_string`
- return the concatenated string `joined_string` of the word list `self.words`

Design of `my_index()` core function

- define `my_index()` method and `self` is the current instance of the class `Sentence`
- `a_word`, a word whose index is to be found in the list is the parameter passed by the calling function
- `self.sentence` is an instance variable of the class `Sentence` and is initialized to a string in the constructor, which is passed as an argument when the class instance is created
- call the method `my_split()` to split `self.sentence` and the result list will be stored in `self.words` (this is the functionality of `my_split()`)

- initialize the variable `index` to zero
- for each `word` in `self.words`, do the below steps:
    - check if `word` is same as `a_word`
    - if both are same words then `return` the `index`
    - else increment the `index` by 1
- return `None` (if flow of execution is out of the loop then it means, element is not found and return inside the loop is not executed)

Design of `my_pop()` core function

- define `my_pop()` method and `self` is the current instance of the class `Sentence`
- `index`, a word at the index `index` is to be found in the list is the parameter passed by the calling function
- `self.sentence` is an instance variable of the class `Sentence` and is initialized to a string in the constructor, which is passed as an argument when the class instance is created
- call the method `my_split()` to split `self.sentence` and the result list will be stored in `self.words` (this is the functionality of `my_split()`)
- to a new variable called `words_length` assign the length of the `self.words` using `len()`
- if the value of `index` is positive and greater than or equal to `words_length` then:
    - `return None`, this is `Index out of bound`
- else if the value of `index` is negative and the difference of `words_length` and the `index` is greater than 2 times `words_length` then:
    - this is also a case of `Index out of bound` so `return None`
- if the above two conditions fail then it implies that the `index` is a legal index
- if `index` is negative then:
    - add the `words_length` to the index value, this makes it easy to retrieve the element
- now store the value of `self.words[index]` in a string variable `word_at_index`
- update `self.words` to remove the element at `index` using following code:
    - using list `slicing`, assign values of `self.words` from index position 0 to the `index` and `index+1` till the last to `self.words`
- return the value `word_at_index`