CSE101

DUE NOV. 23

23:59 KST

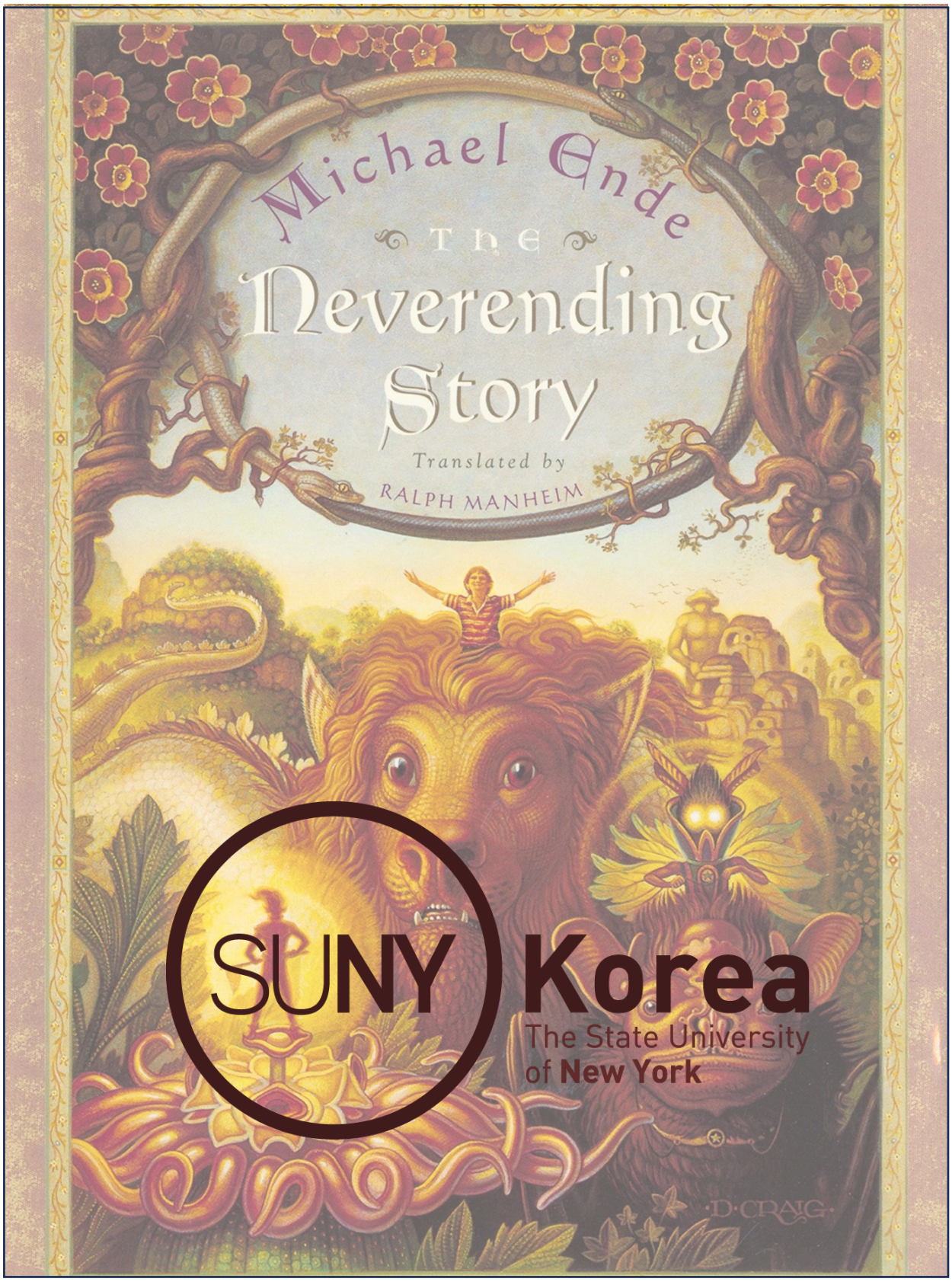
ASSIGNMENT IV

PROFESSOR

FRANCOIS

RAMEAU

Recursion



General instructions

- Please add your name and email in the dedicated location at the top of your code
- Do not use any external library except when explicitly requested
- Try to follow the naming convention proposed by PEP-8 seen in class
- Use ONLY what we have already seen in class. If not, you will get ZERO points
- Use meaningful names for your variables and functions
- If you face a problem submitting your code via GitHub, please contact the professor and the TA by email
- Note that the received code will be tested on a classifier to detect the potential usage of Large Language Models. We will also pay particular attention to plagiarism
- Leave comments in your code to explain your code and describe the difficulties you faced

INVITATION LINK

https://classroom.github.com/a/mOLfm0zx

In this homework, you will take on the role of a developer working for a renowned book publisher. Your boss is eager to ride the wave of AI innovation and has decided to start training neural networks to autonomously write new books! The vast database at the publisher's disposal is ideal for deploying such an application. However, before a Large Language Model can be trained on raw text data, significant preprocessing is required. In this assignment, you will develop several solutions to preprocess text data (e.g., automatically separating each token/word), extract statistics from text (such as occurrences of a specific character or word), and more!

To further challenge you, your boss has stipulated that everything must be implemented using **recursion** exclusively. Loops are not allowed for this assignment, and their use will result in zero points for the task.

Exercise 1: Character frequency counter (1 points)

The very first task you have to solve is to create a function `character_frequency()` which can count the number of occurrences of a character in a string. To be clear, we want to recreate the exact behavior of the Python method `.count()` but using recursion only!

- ***Function***: `character_frequency(text, char)`
- ***Objective***: To count how many times a specific character appears in a given text.
- **Explanation**: This function takes a string (`text`) and a character (`char`), and recursively counts the occurrences of char in text. One way to solve it, consists in splitting the string into the first character and the rest, checking if the first character matches the char, and accumulating the count.
- ***Example***: `character_frequency("alligator", "l")` should return 2, as "l" appears twice in "alligator".

Exercise 2: Text separator (2 points)

Your second goal is more complex, you will create a function `separate_words(),` which takes a continuous string of text and separates it into individual words using spaces as delimiters. The output should be a list containing each individual word of the sentence. This time your function should have the same behavior as the method `.split()`. This task must also be accomplished by employing recursion!

- ***Function:*** `separate_words(text, words, current_word="")`
- ***Objective:*** To separate a string into words based on spaces.
- ***Explanation:*** The function splits a given text into words, using space as the delimiter, and stores the words in the list `words`. It keeps building `current_word` until a space is found, then adds it to the list `words`, and continues recursively.
- ***Example:*** `separate_words("straight road do not make skillful drivers", [])` should produce ["straight", "road", "do", "not", " make", "skillful", "drivers"].

Exercise 3: Word finder (1 points)

Now that you managed to implement the text separator function, it makes it easy to find the index of a specific word in a sentence. This is precisely what you will implement. To be very clear, it is similar to the linear search strategy you have studied before in this class but using recursion. The behavior should be the same as the Python method `.index()`

- ***Function:*** `word_finder(words, query_word, index)`
- **Objective:** To find the index of a specific word in a list of words.

- ***Explanation:*** Starting from the given `index`, this function searches for `query_word` in `words`. If found, it returns the index; otherwise, it recursively checks the next word. Finally, if the query word is not in the list, then return `False`.
  Note that the input `words` is a list of strings obtained by your function `separate_words()`. If you failed to implement this function before, then use the Python `method .split()` instead.
- ***Example:*** word_finder(["the", "little", "cat"], "little", 0) should return 1, as "little" is at index 1.

Exercise 4: Unique word extractor (2 points)

Now, you will construct a new recursive function called `unique_words()` which will delete the duplicate words in the list of words.

- ***Function***: `unique_words(words, unique_words_list)`
- ***Objective***: To extract a list of unique words from a given list of words.
- ***Explanation***: This function takes a list of words and recursively processes it to create a list of unique words. Each word is checked against the `unique_words_list`, and if it isn't already present, it's added. This process continues until all words have been checked. You can also implement this function by deleting duplicate words.
- ***Example***: `unique_words(["apple", "banana", "apple", "pear"], [])` should return ["apple", "banana", "pear"], as these are the unique words in the order they appear.

Exercise 5: Word frequency counter (1 points)

Now that you managed to implement the text separator function, you can easily count the occurrence of a single word in a sentence. This functionally will be implemented in the function called `word_frequency()`.

- ***Function:*** `word_frequency(words, word)`
- ***Objective:*** To calculate the frequency of a specific word in a list of words.
- ***Explanation:*** Similar to `character_frequency`, but this time it counts the occurrences of a whole word in a list. Note that the input `words` is a list of strings obtained by your function `separate_words()`. If you failed to implement this function before, then use the Python `method .split()` instead.
- ***Example:*** `word_frequency(["the", "little", "cat", "the", "the", "the"], "the")` should return 4, as "the" appears four times.

Exercise 6: Create a word histogram (3 points)

Here is the final exercise of this homework. A very powerful feature that can be extracted from a text is its histogram of words, or more specifically, the frequency of each word composing it. This is, for instance, the essence of the famous machine learning technique “Bag of Words”

- **Function**: `word_histogram(words, unique_words_list, histogram)`
- ***Objective***: To create a histogram (a frequency distribution) that counts the occurrences of each unique word in a list.
- ***Explanation***: Leveraging the `unique_words_list` obtained from Exercise 4, this function will recursively go through each word and utilize the `word_frequency` function to determine the number of times each unique word appears in the original words list. These counts are then added to the histogram list, which effectively becomes a list of frequency counts corresponding to the `unique_words_list`.
- ***Example***: Suppose you have already created a list of words and a list of unique words from a text. Your list of words is `["apple", "banana", "apple", "pear", "apple", "banana"]`, and the corresponding list of unique words is `["apple", "banana", "pear"]`. Using the `word_histogram` function, you want to create a histogram that shows the frequency of each unique word:

```
words = ["apple", "banana", "apple", "pear", "apple", "banana"]
unique_words_list = ["apple", "banana", "pear"]
histogram = []

word_histogram(words, unique_words_list, histogram)

print(histogram)  # Expected output: [3, 2, 1]
```

Special note: In this last exercise, you need your function `word_frequency` and `unique_words` to be implemented properly. **If and only if** you could not manage to implement them before, then use the method `.count()` and implement the function `unique_words` using iteration instead of recursion. However, if you choose this non-recursive method, be aware that you will not receive points for these exercises. This exception is made to ensure that all students can attempt and complete Exercise 6. Nonetheless, we encourage you to solve Exercise 4 recursively to practice and benefit from the full learning experience.