Report on the `capitalize_smartly()` Function

Lucas Mirija RAZAFIMANANTSOA.
Eugene BIGIRIMANA.
Lambert Diko NIYONSHUTI.
Makina PATRICK.

October 8, 2025

Function Overview

The function `capitalize_smartly(text: str) → str` aims to correct common errors in capitalization, spacing, and punctuation in a given input text. It processes the text by trimming spaces, capitalizing appropriately, normalizing spaces, and fixing punctuation spacing and sentence boundaries. The input is a possibly misformatted string, and the output is a cleaner, well-capitalized and punctuated string.

Detailed Explanation of Each Code Block

1. Removing Trailing Spaces

```
text = text.strip()
```

Purpose: Removes leading and trailing whitespace characters from the entire text.
Built-in method: `strip()`
Input type: `str`
Output type: `str` with no leading or trailing whitespace

2. Capitalizing the First Character

```
text = text.capitalize()
```

Purpose: Capitalizes the first character of the entire string and converts the rest of the characters to lowercase. This helps to correct casing within the first word and ensures the text starts with a capital letter.
Built-in method: `capitalize()`
Input type: `str`
Output type: `str` with the first character capitalized and the rest lowercase

3. Removing Extra Spaces Between Words while Preserving Line Breaks

```
text_lines = text.split("\n")
for index, line in enumerate(text_lines):
       line = line.split()
       text_lines[index] = " ".join(line)
text = "\n".join(text_lines)
```

Purpose:
- Split the text by line breaks into a list of lines (`text_lines`).

- For each line, split it by whitespace into words, removing any extra spaces by default.
- Rejoin the words with a single space to normalize spacing.
- Rejoin all lines with "\n" to restore the original line breaks.

Built-in methods and types:

- `split("\n")` on `str` returns `List[str]`.
- `split()` on `str` returns `List[str]` splitting by arbitrary whitespace.
- `join()` on `str` takes `Iterable[str]` and returns `str`.

4. Handling Spaces Around Punctuation

```
punctuations = ".,;!?:"
characters_list = []
i = 0
while i < len(text):
        char = text[i]
        if char in punctuations:
                if characters_list[-1] == " ":
                        characters_list.pop()
                        characters_list.append(char)
                if i+1 < len(text) and text[i+1] != " ":
                        characters_list.append(" ")
        else:
                characters_list.append(char)
        i += 1
```

Purpose:

- Iterate through the characters in the text.
- For punctuation characters defined in `punctuations`, remove any space directly before the punctuation, append the punctuation, and ensure there is exactly one space after it (except at the text end).
- For non-punctuation characters, just append them.

Variable:

- `characters_list`: a list of characters (`List[str]`) used to build the corrected text.
- The loop uses an index `i` to traverse the text string.

5. Capitalization and Punctuation Adjustment After Sentence Endings

```
line_skip_indexes = []

for index, value in enumerate(characters_list):

        if value in ".!?":
                if index+2 < len(characters_list) and characters_list[index+2].isalpha():
                        characters_list[index+2] = characters_list[index+2].upper()

        condition_I = (value == "i") and (characters_list[index+1] in " ,\"'") and (
            characters_list[index-1] in " ,\"'")
        if index+1 < len(text) and condition_I:
                characters_list[index] = characters_list[index].upper()

        if index+1 < len(text) and value == '\n':
                characters_list[index+1] = characters_list[index+1].upper()

                last_char = characters_list[index - 1]
                if last_char == ' ' and index - 2 >= 0:
                        last_char = characters_list[index - 2]

                if last_char not in punctuations:
                        line_skip_indexes.append(index)
```

```
            line_skip_indexes.reverse()
            for i in line_skip_indexes:
                characters_list.insert(i, '.')
```

Purpose:

- Capitalize the first alphabetical character two positions after sentence-ending punctuation marks (`.`, `!`, `?`). The +2 is for the punctuation character and the space that follows.
- Capitalize standalone lowercase `"i"` when surrounded by spaces or common punctuation to convert it to `"I"`.
- Capitalize the first character after newline characters.
- Check if each line ends with proper punctuation; if not, insert a period at the line break.

Built-in method:
- `isalpha()` on `str` returns `bool`, used to check if a character is alphabetic.
Notes on indexing:
- The code checks boundaries to avoid index errors.

6. Final fixes and Return

```
            text = "".join(characters_list)
            text_lines = text.split('\n')
            for index, line in enumerate(text_lines):
                text_lines[index]= line.strip()
            text = "\n".join(text_lines)

            if text[-1] not in punctuations:
                text = text +'.'
            return text
```

Purpose: Remove trailing spaces at each end of lines and add punctuation at the very end of the entire text.
Built-in method: `join()` takes `List[str]` returns `str`.

Algorithms Summary

Algorithm: Fixing Spaces Around Punctuation

- Initialize an empty list `characters_list`.
- For each character `char` in the input text:
  - If `char` is punctuation:
    - If last character in `characters_list` is space, remove it.
    - Append `char`.
    - If next character exists and is not space, append a space.
  - Else, append `char`.

Algorithm: Capitalization and Punctuation Fix

- Iterate over `characters_list` with index `i`.
- When current char is `'.'`, `'!'`, `'?'` and there is a letter two chars after, capitalize that letter.
- Capitalize standalone lowercase `'i'` when surrounded by spaces or punctuation.
- When newline `'\n'` is found, capitalize the following character.
- Collect line break indexes where punctuation is missing and insert period before new line.

Summary of Built-in Methods and Their Outputs

Method	Input	Output
`strip()`	`str`	`str` without leading/trailing whitespace
`capitalize()`	`str`	`str` with first character uppercase, rest lowercase
`split("\n")`	`str`	`List[str]` split on newline
`split()`	`str`	`List[str]` split on whitespace
`join()`	`Iterable[str]`	`str` joined by delimiter
`isalpha()`	`str (length 1)`	`bool` if character is alphabetic

Remark on Proper Noun Capitalization

Although the project specification explicitly requested **no external libraries**, the proper noun capitalization functionality using `spaCy` was added as an optional enhancement. This addition is not required for the core project objectives; it serves as an extra improvement to demonstrate how machine learning models can assist in more precise text formatting and capitalization.

Extra: Proper Noun Detection and Capitalization with spaCy

Overview

In addition to general capitalization and punctuation corrections, the program can include a complementary mechanism to identify and correctly capitalize **proper nouns** (e.g., names of people, places, organizations). This is done through two key functions:

- `is_proper_noun(word: str)` → `bool`
- `capitalize_PN(text: str)` → `str`

These functions rely on the **spaCy** Natural Language Processing (NLP) library. Two models are used simultaneously:

- `en_core_web_sm`: English-specific model with part-of-speech (POS) tagging.
- `xx_ent_wiki_sm`: Multilingual entity recognition model based on Wikipedia.

Combining these two models allows the code to capture a broader range of proper nouns, including non-English ones.

1. Detecting Proper Nouns

```
import spacy

# Loading the models
nlp_en = spacy.load("en_core_web_sm")
nlp_xx = spacy.load("xx_ent_wiki_sm")

def is_proper_noun(word):
        # Label the word using both models
        doc_en = nlp_en(word)
        doc_xx = nlp_xx(word)

        for token in doc_en:
                if token.pos_ == "PROPN":
                        return True
                else:
                        return False

        for token in doc_xx:
                if token.pos_ == "PROPN":
                        return True
        else:
                return False
```

Purpose: This function determines whether a given word should be capitalized as a proper noun by applying part-of-speech tagging. In spaCy, the POS tag `PROPN` stands for *proper noun*.

Step-by-step process:

1. The word is processed by both NLP models (`doc_en` and `doc_xx`) and turned into tokens.
2. For each token, the function checks if its POS tag equals `"PROPN"`.

Methods used:

- `nlp(word)`: returns a `Doc` object after linguistic analysis.
- `token.pos_`: returns the part-of-speech tag of the token.

2. Capitalizing Proper Nouns in Text

```
def capitalize_PN(text):
        lines = text.split('\n')
        sentence = []
        for index, line in enumerate(lines):
                sentence.append(line)
                words = sentence[index].split()
                for i, w in enumerate(words):
                if is_proper_noun(w):
                words[i] = w.capitalize()
                l = " ".join(words)
                sentence[index] = l
        text = "\n".join(sentence)
        return text
```

Purpose: This function applies `is_proper_noun` to each word in the text. If a word is identified as a proper noun, it is capitalized using Python's built-in `capitalize()` method.

Step-by-step process:

1. Split the text into lines to preserve original line breaks.
2. For each line, split it into words.
3. Apply `is_proper_noun` to each word.
4. If the function returns `True`, capitalize the word.
5. Reassemble the line and then the entire text.

Built-in methods used:

- `split()` and `join()` for word and line processing.
- `capitalize()` to ensure the first letter of proper nouns is uppercase.

Integration with `capitalize_smartly`: The text is first passed through `capitalize_smartly` to correct punctuation and sentence casing. Then, it is passed through `capitalize_PN` to handle proper noun capitalization using linguistic analysis. This two-step pipeline ensures that both grammatical and semantic capitalization rules are applied.

3. Example

```
text = """ Hello eveRyone.
my name Is lucas and this is kenny
how are you?personnaly , i feeL goOd !"""

print("Before capitalization:")
print(text)

text = capitalize_PN(capitalize_smartly(text))

print("After capitalization:")
print(text)
```

Output:

```
Before capitalization :
Hello   eveRyone.
my name Is mirija and this is kenny
```

```
how are you?personnaly  , i feeL goOd !

After capitalization :
Hello everyone.
My name is Mirija and this is Kenny.
How are you? Personnaly, I feel good!
```

Here, both `Mirija` and `Kenny` were detected and capitalized as proper nouns by spaCy, while other words were normalized through `capitalize_smartly()`.