Text Analyser

A TERMINAL APPLICATION

BY JOHN FABER RODRIGUEZ CORREDOR

T1A3

2023

analizer.py

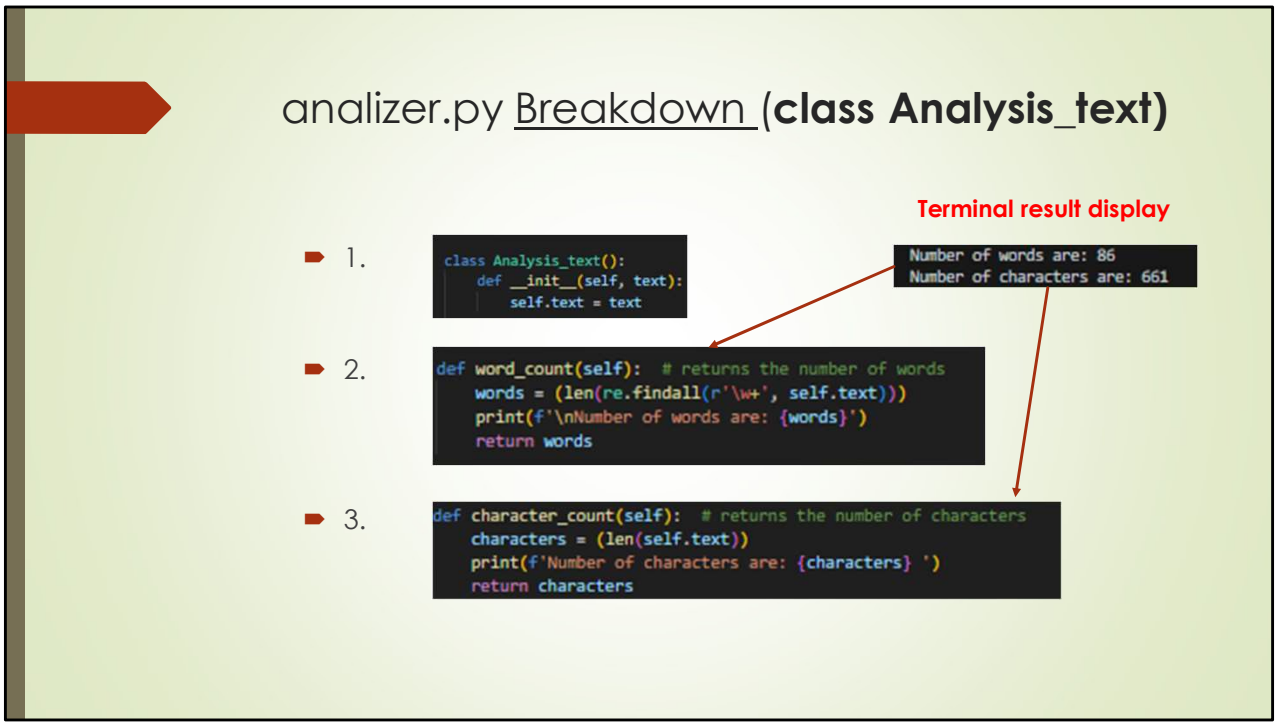
```python
class Analysis_text():
    def __init__(self, text):
        self.text = text

    def word_count(self):  # returns the number of words
        words = (len(re.findall(r'\w+', self.text)))
        print(f'\nNumber of words are: {words}')
        return words

    def character_count(self):  # returns the number of characters
        characters = (len(self.text))
        print(f'Number of characters are: {characters} ')
        return characters

    def lines_count(self):  # returns the number of lines excluding empty lines
        line_count = len([line for line in self.text.splitlines() if line.strip()])
        print(f'Number of lines are: {line_count}')  # list comphrension
        return line_count

    def paragraph_count(self):  # returns the number of paragraphs
        paragraphs_count = len([paragraph for paragraph in self.text.split('\n\n') if paragraph.strip()])
        print(f'Number of paragraphs are: {paragraphs_count}')  # list comphrension
        return paragraphs_count

    def language_text(self):  # returns the language text
        language = (detect(self.text)).upper()
        print(f'language text: {language}')
        return language
```

```python
# feature #2 Word frequency analysis


class Word_Frequency():
    def __init__(self, text):
        self.text = text

    def word_frequency(self):  # will return top 5 most common words
        frequency = dict(Counter(self.text.split()).most_common(5))
        print(f'\nyour top 5  most common words are: \n\n {frequency}')

        return frequency

# feature #3 keyword extraction


class Keyword():
    def __init__(self, text):
        self.text = text

    def keyword_extraction(self):
        language = (detect(self.text))
        language_list = {'en': 'english', 'es': 'spanish', 'fr': 'french', 'it': 'italian', 'de': 'german'}
        rake_language = language_list.get(language, 'english')  # 'Get' used to extract from dictionaries
        rake_nltk_var = Rake(language=rake_language)
        rake_nltk_var.extract_keywords_from_text(self.text)
        keyword_extracted = rake_nltk_var.get_ranked_phrases()[:5]
        print(f'\nThe Top 5 keywords of your text are:\n\n {keyword_extracted}')
        return keyword_extracted
```

- The analyzer.py script provided contains three classes (Analysis_text, Word_Frequency, and Keyword), each encapsulating different text analysis features.

<u>Highlights of analizer.py</u>

- Each class focuses on a specific functionality, making the codebase more **<u>readable, understandable, and maintainable</u>**.
- **Reusability:** if another Python script or application requires similar text analysis capabilities, developers can directly import and use the classes from analyzer.py
- If there is a need to add more text analysis features in the future, developers can extend the analyzer.py file by adding new classes or methods **<u>without altering existing functionalities</u>**
- analyzer.py first allows to establish **the foundation** and the functionality that main.py builds upon .



1. Class Analysis_text:

- **def __init__(self, text):**: This is the constructor method of the class.
- The **__init__** method is automatically called when a new object of the class is created.
- **self.text = text** the value passed as the text parameter is assigned to the instance variable self.text
- **self** refers to the instance of the class, allows access to variables and methods within the class.

2. word_count method :

- It calculates the number of words in the input text stored in the instance variable self.text
- **r'\w+' (Regular Expression (Regex) Pattern)** is used to match one or more-word characters, alphanumeric character plus underscore '_', and specifies that there should be one or more occurrences of the prior pattern.
- **re.findall(pattern, string)** is a function from the re module in Python that finds all occurrences of the specified pattern in the given string.

- **len(re.findall(r'\w+', self.text))** calculates the number of words by finding the length of the list of words.
- **print(f'\nNumber of words are: {words}’)** prints the number of words to the console.
- The **f-string** format is used to embed the value of words into the string for printing.
- **return words** returns the number of words. This allows the calling code to capture the word count and use it.

3. Character_count method:

- Calculates the number of characters in the input text stored in the instance **variable self.text**.
- **len(self.text)** calculates the length of the input text stored in self.text.
- **print(f'Number of characters are: {characters} ')** prints the number of characters to the console.
- **return characters** returns the number of characters. This allows the calling code to capture the character and use it

analizer.py Breakdown (**class Analysis_text**)

4.
```python
def lines_count(self):  # returns the number of lines excluding empty lines
    line_count = len([line for line in self.text.splitlines() if line.strip()])
    print(f'Number of lines are: {line_count}')  # list comphrension
    return line_count
```

5.
```python
def paragraph_count(self):  # returns the number of paragraphs
    paragraphs_count = len([paragraph for paragraph in self.text.split('\n\n') if paragraph.strip()])
    print(f'Number of paragraphs are: {paragraphs_count}')  # list comphrension
    return paragraphs_count
```

6.
```python
def language_text(self):  # returns the language text
    language = (detect(self.text)).upper()
    print(f'language text: {language}')
    return language
```

```
Number of lines are: 1
Number of paragraphs are: 1
language text: DE
```

Terminal result display

4. **lines_count** method:
- Calculates the number of lines in the input text excluding empty lines.
- **self.text.splitlines()** splits the input text into a list of lines.
- **len[line for line in self.text.splitlines() if line.strip()]** is a list comprehension. Calculates the length of the list of non-empty lines, giving the number of lines excluding empty lines.
- line.strip() checks if the line, after stripping whitespace characters, is non-empty.
- **print(f'Number of lines are: {line_count}')** prints the number of lines (excluding empty lines) to the console.
- return line_count returns the number of lines (excluding empty lines). This allows the calling code to capture the lines count and use it.

5. paragraph_count method:
- Calculates the number of paragraphs in the input text.
- **len([paragraph for paragraph in self.text.split('\n\n') if paragraph.strip()])** calculates the length of the list of non-empty paragraphs, giving the number of paragraphs excluding empty paragraphs.
- **print(f'Number of paragraphs are: {paragraphs_count}')** prints the number of paragraphs (excluding empty paragraphs) to the console.

- return paragraphs_count returns the number of paragraphs (excluding empty paragraphs). This allows the calling code to capture the paragraph count and use it.

6. **language_text** method:
- Detects and returns the language of the input text
- **(detect(self.text))** calls the detect function from the **langdetect library** to determine the language of the input text.
- **upper()** converts the detected language string to uppercase.
- **print(f'language text: {language}')** prints the detected language (in uppercase) to the console.
- **return language** returns the detected language in uppercase. This allows the calling code to capture the detected language and use it .

analizer.py Breakdown (**class Word_Frequency**)

1.
```python
class Word_Frequency():
    def __init__(self, text):
        self.text = text
```

2.
```python
    def word_frequency(self):  # will return top 5 most common words
        frequency = dict(Counter(self.text.split()).most_common(5))
        print(f'\nyour top 5  most common words are: \n\n {frequency}')

        return frequency
```

```
your top 5  most common words are:

 {'und': 4, 'für': 3, 'seine': 3, 'Deutschland': 2, 'ist': 2}
```

Terminal result display

1. Class Word_Frequency:

- **def __init__(self, text):**: This is the constructor method of the class.
- The **__init__** method is automatically called when a new object of the class is created.
- **self.text = text** the value passed as the text parameter is assigned to the instance variable self.text
- **self** refers to the instance of the class, allows access to variables and methods within the class.

2. **word_frequency** method
- Calculates the frequency of words in the input text and returns the top 5 most common words along with their counts.
- **Counter(self.text.split())** creates a **Counter object**, which is a list of tuples, to count the occurrences of each word in the list of words.
- **.most_common(5)** returns a list of the 5 most common words along with their counts from the **Counter object**.
- **dict()** converts this list of tuples into a dictionary where the **words are keys**, and their **counts are values**. Very useful for testing purposes.

- **print(f'\nyour top 5 most common words are: \n\n {frequency}')** prints the top 5 most common words and their counts to the console.
- **return frequency** returns the dictionary containing the top 5 most common words and their counts.
- **return frequency** allows the calling code to capture the word frequency dictionary and use it .

analizer.py Breakdown (**class Word_Frequency)**

1.
2.

```python
class Keyword():
    def __init__(self, text):
        self.text = text

    def keyword_extraction(self):
        language = (detect(self.text))
        language_list = {'en': 'english', 'es': 'spanish', 'fr': 'french', 'it': 'italian', 'de': 'german'}
        rake_language = language_list.get(language, 'english')  # 'Get' used to extract from dictionaries
        rake_nltk_var = Rake(language=rake_language)
        rake_nltk_var.extract_keywords_from_text(self.text)
        keyword_extracted = rake_nltk_var.get_ranked_phrases()[:5]
        print(f'\nThe Top 5 keywords of your text are:\n\n {keyword_extracted}')
        return keyword_extracted
```

```
The Top 5 keywords of your text are:

 ['deutsche küche legt großen wert', 'reiche kulturelle geschichte', 'internationalen köstlichkeiten reicht', 'exzellente küche bekannt', 'zieht besucher']
```

Terminal result display

1. **Class Keyword:**

- **def __init__(self, text):**: This is the constructor method of the class.
- The **__init__** method is automatically called when a new object of the class is created.
- **self.text = text** the value passed as the text parameter is assigned to the instance variable self.text
- **self** refers to the instance of the class, allows access to variables and methods within the class.

2. **keyword_extraction:**
- (detect(self.text)) calls the **detect function** from the **langdetect library** to determine the language of the input text.
- **language_list** is a dictionary that maps detected languages to their corresponding RAKE language codes.
- **rake_language = language_list.get(language, 'english')** uses the get method to extract the RAKE language code based on the detected language. It defaults to 'english'.
- **rake_nltk_var = Rake(language=rake_language)** creates an instance of the Rake

class from the rake_nltk library. The language parameter is set to the determined RAKE language code.
- **rake_nltk_var.extract_keywords_from_text(self.text)** processes the input text and extracts keywords using the RAKE algorithm.
- **rake_nltk_var.get_ranked_phrases()[:5**] retrieves the top 5 keywords from the extracted phrases using **slicing ([:5]).**
- **print(f'\nThe Top 5 keywords of your text are:\n\n {keyword_extracted}')** prints the top 5 keywords to the console.
- **return keyword_extracted** returns the list of top 5 keywords.
- This allows the calling code to capture the extracted keywords and use them if needed.

analizer.py Breakdown **(Modules and libraries)**

```
import re
from langdetect import detect
from collections import Counter
from rake_nltk import Rake
import nltk
nltk.download('stopwords')
nltk.download('punkt')
```

1. **re** python module, used for text processing and pattern matching.
2. **Langdetect** library for language detection. **Detect** function used to detect languages of text.
3. **Collections** module for built-in types. **Counter** dictionary subclass for counting hashable objects.
4. **rake_nltk** provides **rake** (rapid Automatic Keyword Extraction)
5. **Nltk** (Natural Language Toolkit) library for human language data.
6. **Nltk.download('stopwords')** stopwords are common words such as 'and', 'the', 'is', and more. Used during text processing.
7. **Nltk.download('punkt')** punkt is a tokenizer for tokenizing words and sentences. It's a process for breaking text into words and sentences.

main.py

```python
from analyzer import Analysis_text
from analyzer import Word_Frequency
from analyzer import Keyword


def analyze_text(analyzer):  # Feature #1
    analyzer.word_count()
    analyzer.character_count()
    analyzer.lines_count()
    analyzer.paragraph_count()
    analyzer.language_text()


def get_user_input():  # input text from user
    while True:
        try:
            user_text = input('Enter the text you want to analyze:\n\n ')
            if len(user_text.strip()) <= 10:
                raise ValueError('Input cannot be empty. Minimum 10 words')
            return user_text
        except Exception as e:
            print(f"Error: {e}")
```

```python
def main():
    print('----------------------------')
    print('Welcome to Text File Analyzer')
    print('----------------------------')
    user_text = get_user_input()

    while True:
        analyzer = Analysis_text(user_text)
        frequency = Word_Frequency(user_text)
        keywords = Keyword(user_text)

        print('\nChoose an option for you text\n')
        print('1. Text Analysis')
        print('2. Word Frequency Analysis')
        print('3. Keywords extraction')
        print('4. Exit')

        choice = input('\nEnter choice:  ')

        try:
            if choice == '1':
                analyze_text(analyzer)
            elif choice == '2':
                frequency.word_frequency()

            elif choice == '3':
                keywords.keyword_extraction()

            elif choice == '4':
                print('--------------------------------')
                print('Thanks for using Text Analizer! ')
                print('--------------------------------')
                break
            else:
                raise ValueError('Invalid choice, Input must be 1,2,3 or 4.')
            while True:
                option = input('\nWould you like to continue? yes/no ').lower()
                if option in ['yes', 'no']:
                    break
                else:
                    print("\nInvalid choice, Input must be 'yes' or 'no'")
            if option != 'yes':
                print('--------------------------------')
                print('Thanks for using Text Analyzer!')
                print('--------------------------------')
                break
        except Exception as e:
            print(f"Error: {e}")


if __name__ == "__main__":
    main()
```

Main.py Highlights

- Utilizes classes from analyzer.py
- Utilizes methods from analyzer.py
- Gets inputs from users
- Instances created so user can perform different text analysis
- Menu was created for text analysis, word frequency analysis, keyword extraction, and exiting the program.
- The code contains error handlers that catches exceptions and displays error messages
- Implementation of loops for a constant user engagement with the app
- Welcome and thanks messages added for graceful user experience
- main.py interacts with analyzer.py to perform text analysis tasks by importing classes from it.
- code provides an interactive involvement user-app

main.py Breakdown

1.
```python
from analyzer import Analysis_text
from analyzer import Word_Frequency
from analyzer import Keyword
```

- Import classes from analyzer.py

2.
```python
def analyze_text(analyzer):  # Feature #1
    analyzer.word_count()
    analyzer.character_count()
    analyzer.lines_count()
    analyzer.paragraph_count()
    analyzer.language_text()
```

- Function takes **analyzer** as a parameter to perform all **methods** in **Analysis_text** class.
- It executes the basic text analysis .

3.
```python
def get_user_input():  # input text from user
    while True:
        try:
            user_text = input('Enter the text you want to analyze:\n\n ')
            if len(user_text.strip()) <= 10:
                raise ValueError('Input cannot be empty. Minimum 10 words')
            return user_text
        except Exception as e:
            print(f"Error: {e}")
```

- Function gets user input.
- Prevents empty inputs by raising exceptions(**try, except, Exception)** with messages of errors .
- **return user_text** returns the valid non-empty input provided by the user to the calling code.
- **while True** loop validates that user input is not empty, executed the function after validation.

main.py Breakdown

1. **While True** ensure app is running until user exit.
2. **Instances** created with **user_text** input which anew in every iteration of the loop.
3. Print menu for text analysis options (1, 2, 3, or 4 exit)
4. User selection stores in **choice** variable
5. **if-elif-else** statements used to execute user's selection.

```python
def main():
    print('----------------------------')
    print('Welcome to Text File Analyzer')
    print('----------------------------')
    user_text = get_user_input()

    while True:
        analyzer = Analysis_text(user_text)
        frequency = Word_Frequency(user_text)
        keywords = Keyword(user_text)

        print('\nChoose an option for you text\n')
        print('1. Text Analysis')
        print('2. Word Frequency Analysis')
        print('3. Keywords extraction')
        print('4. Exit')

        choice = input('\nEnter choice:  ')

        try:
            if choice == '1':
                analyze_text(analyzer)
            elif choice == '2':
                frequency.word_frequency()

            elif choice == '3':
                keywords.keyword_extraction()

            elif choice == '4':
                print('--------------------------------')
                print('Thanks for using Text Analizer! ')
                print('--------------------------------')
                break
            else:
                raise ValueError('Invalid choice, Input must be 1,2,3 or 4.')
            while True:
                option = input('\nWould you like to continue? yes/no ').lower()
                if option in ['yes', 'no']:
                    break
                else:
                    print("\nInvalid choice, Input must be 'yes' or 'no'")
            if option != 'yes':
                print('--------------------------------')
                print('Thanks for using Text Analyzer!')
                print('--------------------------------')
                break
        except Exception as e:
            print(f"Error: {e}")
```

6. Choice #1 **analyze_text** function is called with the **analyzer object**.
7. Choice #2 **word_frequency** function is called with the **frequency object**.
8. Choice #3 **keyword_extraction** function is called with the **keywords object**.
9. Choice #4 farewell message is printed, and the **loop is terminated** with the **break** statement.
10. After the selection analysis, app prompts if user wishes to continue. **Yes** the loop continues and **No**

farewell message is printed, and the **loop is terminated** with the **break** statement

11. Wrong answers or invalid choices are handled by raising exceptions(**try, except, Exception)** with messages of errors .

main.py Breakdown

```python
if __name__ == "__main__":
    main()
```

- This block of code is checking whether main.py is being run directly or imported into another script.
- **__name__** variable is set to "**__main__**", stablishing the actual script is not imported , as result the **main()** function will be executed.
- Ensures **main()** has a clear path to be the entry point for the app.
- The contrary scenario will be that **main()** will be skipped as the actual module is not being run directly as a script.
- This construct in python provides modularity and reusability.
- Unintended execution is prevented when the file imported as a module
- It provides a clean structure within python programs.

Learnings

- Better understanding of how to use OOP, classes, methods.
- Improved in how to use looping and flow control.
- Improved in File Structure and Modularity.
- Error handling makes an app run more elegant.
- Libraries make life easier, do the heavy lifting for a lot of complex processes.
- Resources can be found online even I joined Stackoverflow community
- PEP 8 helped me to understands in an atomic level how a piece of code can be cleaned for better performance , presentation, and readability.
- The number of things that can be built using Python are endless.
- How to structure a project.