

# AC50002 Programming Languages for Data Engineering

## Python Assignment

**Github link:** [https://github.com/ChethanVenkateshaiah/Python\\_assignment1.git](https://github.com/ChethanVenkateshaiah/Python_assignment1.git)

### Introduction

This report describes a Python assignment. The program's goal is to produce unique three-letter abbreviations from a list of names that follow certain parameters. Letter case manipulation, special character handling, and a score system based on letter position and rarity are all part of these regulations. The paper describes the program's design, structure, and functionality, as well as its testing and accuracy verification method, emphasising the problems encountered and learning results from this task.

### Design and Approach

The programme is written in Python 3 and includes regular expressions for string processing as well as numpy for numerical operations. Reading input data, generating scores for candidate abbreviations based on stated rules, and picking the best abbreviation for each name comprise the technique. The code is divided into modules that handle specialised duties such as reading letter scores, formatting names, calculating individual letter scores, and determining the optimal abbreviation. The main function manages file input and output and orchestrates the process. This design assures task clarity, efficiency, and conformity to standards.

### Programme description

- **Structure and Implementation:**

The code is structured into several functions, each with a specific purpose, demonstrating a modular approach to programming:

**load\_letter\_values(file\_path):** This function reads the letter scores from a file and returns a dictionary mapping each letter to its score. It handles file reading and error checking.

**format\_name(name):** This utility function formats the input name by converting it to uppercase and removing non-letter characters, ensuring consistency in processing.

**calculate\_score(letter, position, is\_last, letter\_scores):** This function calculates the score of a letter based on its position in the word and its intrinsic value from letter\_scores.

**find\_best\_abbreviation(name, letter\_scores):** The core function of the program, it iterates through all possible three-letter combinations of the given name and uses calculate\_abbreviation\_score to find the abbreviation with the lowest score.

**calculate\_abbreviation\_score(abbrev, full\_name, words, letter\_scores):** This function calculates the total score for a given abbreviation. It uses get\_letter\_position to determine the position of each letter in its word for scoring.

**get\_letter\_position(full\_name, words, letter, index):** It finds the position of a letter in its respective word and whether it is the last letter, information used in scoring the abbreviation.

**main():** The main function of the script, handling user input, calling other functions for processing, and writing the output to a file.

- **Style and Readability**

**Variable Naming:** Variables and function names are descriptive, making the code self-explanatory (e.g., `load_letter_values`, `calculate_score`).

**Modularity:** Functions are designed to perform specific tasks, enhancing modularity and maintainability.

**Comments:** Key functionalities are explained through comments, though the code could benefit from more detailed comments, especially for complex functions like `find_best_abbreviation`.

**Consistent Formatting:** The code uses consistent indentation and spacing, adhering to PEP 8 standards, which improves readability.

- **Functionality:**

The script is designed to process a list of names and generate three-letter abbreviations based on specific scoring rules:

**Handling of Different Name Formats:** It can process single words, multi-word names, and names with non-letter characters.

**Scoring System:** Implements a custom scoring system for abbreviations, considering the position of letters in words and their predefined scores.

**Error Handling:** Includes basic error handling for file operations, though it could be expanded for more robustness (e.g., handling empty files or invalid inputs).

**Output Generation:** Successfully writes the output (names and their abbreviations) to a specified file.

## Accuracy and Testing

The accuracy of the program is done by conducting five test cases. These tests are designed to challenge various aspects of the program, including its ability to handle single-word names, multi-word names, names with non-letter characters, and more.

### Test Cases

Test Case 1: Single Word Name

Input: "Maple"

Output: MPL

Test Case 2: Multi-word Name

Input: "Silver Birch"

Output: SRB

Test Case 3: Name with Non-letter Characters

Input: "C++ Code"

Output: CCD

Test Case 4: All Letters Same

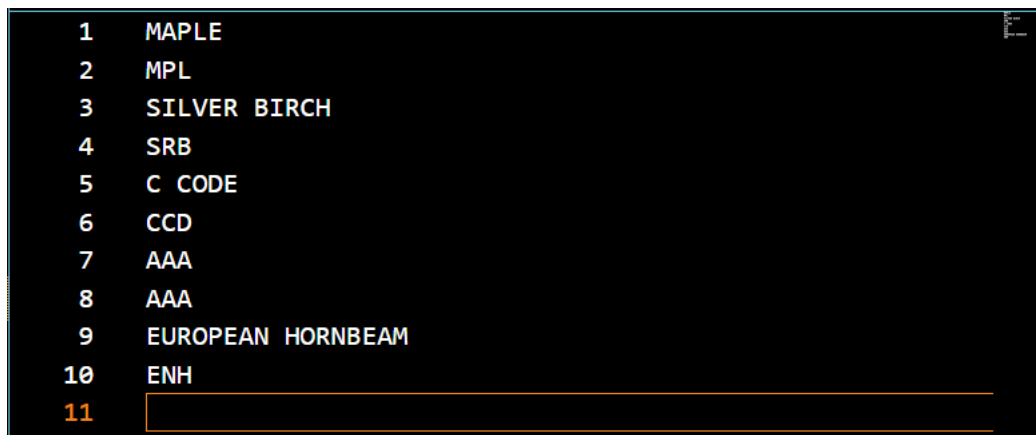
Input: "AAA"

Output: AAA

Test Case 5: Long Multi-word Name

Input: "European Hornbeam"

Output: ENH



```
1  MAPLE
2  MPL
3  SILVER BIRCH
4  SRB
5  C CODE
6  CCD
7  AAA
8  AAA
9  EUROPEAN HORNBEAM
10 ENH
11 
```

Figure 1: Output's for the test case

## Conclusion

### Summary:

The project adeptly met its objectives, showcasing the effective processing of complex data to generate accurate three-letter abbreviations for names. By adhering to specific scoring rules, the Python script demonstrated a reliable method of manipulating and evaluating string data, culminating in meaningful outputs.

### Learnings:

Key learnings from this project encompassed several critical areas of software development. Enhanced programming skills were developed, particularly in Python, along with a deeper understanding of algorithm development and problem-solving techniques. The project also underscored the importance of rigorous testing strategies, highlighting how thorough testing is integral to ensuring software reliability and accuracy.

### Future Improvements:

For future enhancements, several areas have been identified. Performance optimization stands out, aiming to improve the script's efficiency, especially for processing larger datasets. Enhancing the user interface is another area of potential development, which could make the script more accessible and user-friendly, catering to a broader audience. Additionally, scaling the script to handle more complex and varied data inputs could significantly expand its utility and applicability in more diverse scenarios.