Project Report: Console-Based Typing Speed Tester

1. Cover Page

Field	Detail
Project Title	Console-Based Typing Speed Tester
Name	Aryaman singh chauhan
Registration no.	25BCY10217
Technology Used	Python 3.x

2. Introduction

This project is a simple, command-line application designed to measure a user's typing speed, expressed in Words Per Minute (WPM). Built entirely in standard Python, the tool provides a practical, low-overhead way for users to practice their typing skills and receive immediate performance feedback. The core functionality involves displaying a randomly selected target phrase, timing the user's input duration, and calculating the final WPM score based on accuracy and speed.

3. Problem Statement

In an increasingly digital world, typing proficiency is a fundamental skill. The problem addressed by this project is the lack of an accessible, lightweight tool to accurately measure and track typing speed without relying on external web services or complex installations. The goal is to create a reliable and self-contained solution to help users assess and improve their typing efficiency.

4. Functional Requirements

The system must be able to perform the following core functions:

1. **Text Selection:** Randomly select a target sentence from a predefined list of phrases.
2. **Timing:** Start a timer immediately upon presentation of the text and stop it immediately after the user submits their input.
3. **Input Capture:** Accept and process the user's typed input from the console.
4. **Comparison:** Accurately compare the user's input against the target text (ignoring leading/trailing whitespace).
5. **WPM Calculation:** Calculate the Words Per Minute (WPM) using the standard formula: WPM = (Total Characters / 5) / Time (in minutes).
6. **Feedback:** Provide clear, immediate feedback on whether the input was correct or incorrect.
7. **Error Reporting:** In case of an incorrect entry, display the time taken and highlight the mismatched words to aid the user in identifying errors.

5. Non-functional Requirements

1. **Usability:** The interface must be simple, text-based, and intuitive, requiring minimal cognitive load.
2. **Performance:** The WPM calculation and result display should be instantaneous.
3. **Reliability:** The time measurement using the `time` module must be accurate to ensure a reliable WPM score.
4. **Portability:** The code should run without modification on any platform (Windows, macOS, Linux) that has a standard Python 3 interpreter installed, as it relies only on built-in standard libraries (`random`, `time`).

6. System Architecture

The system employs a **Monolithic Single-Script Architecture**.

- **Front-end/Interface:** Command-Line Interface (CLI) using Python's `input()` and `print()` functions.
- **Business Logic:** Contained within the `typing_test()` function, which orchestrates text selection, timing, and result processing.
- **Utility/Calculation Layer:** The `calculate_wpm()` function handles the specific mathematical logic for speed measurement.
- **External Dependencies:** None, as it uses only the standard Python libraries (`random` and `time`).

7. Design Diagrams

Workflow Diagram

The system's flow is linear and highly deterministic, following these steps:

1. **Initialization:** The `typing_test()` function is called.
2. **Text Selection:** A phrase is chosen randomly from the `sentences` list.
3. **Display:** The target text is printed to the console.
4. **Start Timing:** `start_time = time.time()` records the initial timestamp.
5. **Input:** The user is prompted for input via `input()`.
6. **Stop Timing:** `end_time = time.time()` records the completion timestamp, and `total_time` is calculated.
7. **Validation:** The user input is stripped and compared to the target text.
8. **Result Processing:**
    - **If Match:** Call `calculate_wpm()`, print WPM and time taken.
    - **If Mismatch:** Print error, show target vs. user input, and iterate word-by-word to detail the differences.
9. **Termination:** The program ends.

8. Design Decisions & Rationale

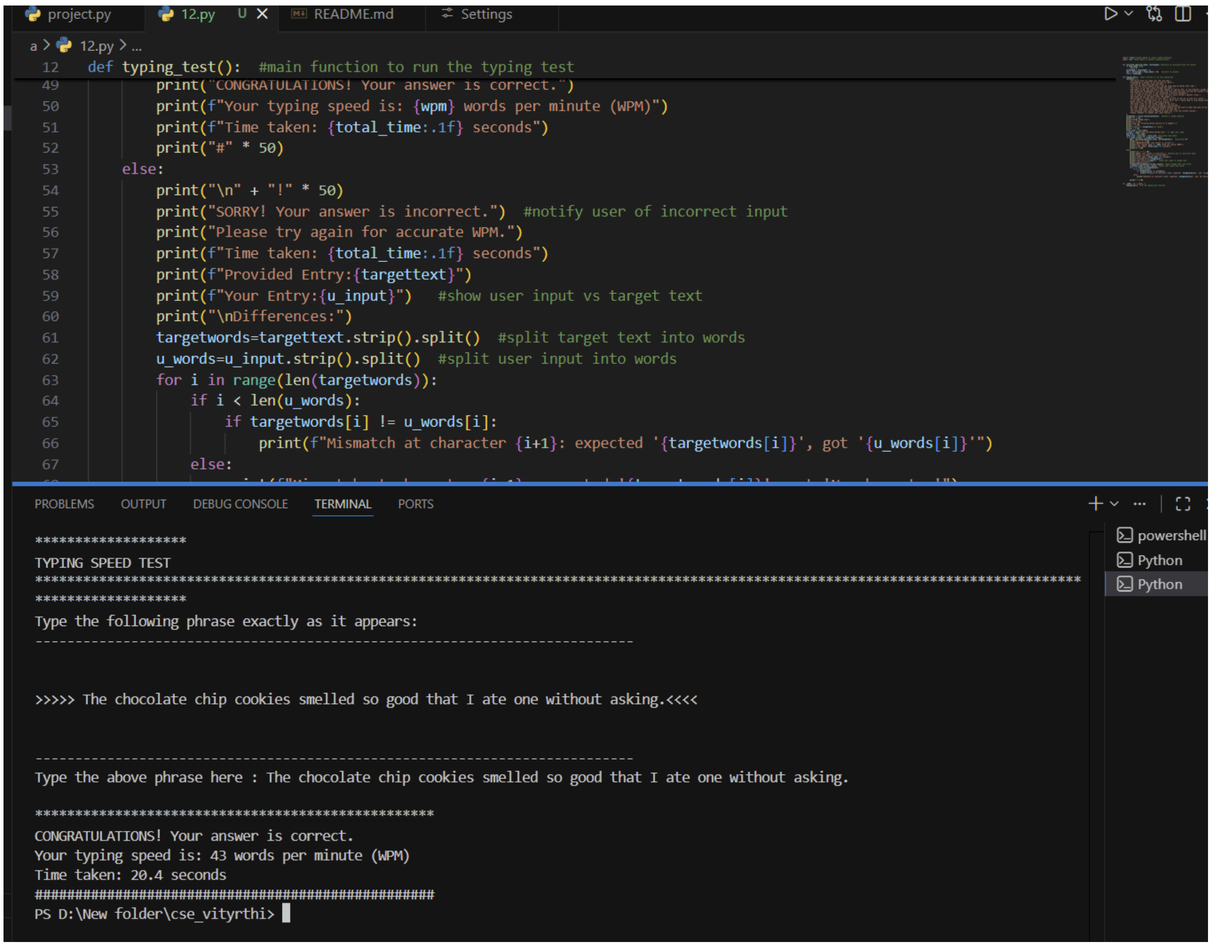
Component	Decision	Rationale
WPM Calculation	`wordstyped = textlength / 5`	This follows the conventional WPM standard, where one "word" is defined as 5 characters.
Timing Mechanism	Use of `time.time()`	`time.time()` provides high-resolution time measurement suitable for short, interactive tasks like a typing test, ensuring accuracy in speed measurement.
Input Validation	Strict comparison (`.strip() == .strip()`)	Requires exact matching to ensure accurate performance metrics. The use of `.strip()` accounts for common errors like accidentally adding leading/trailing spaces.
Error Feedback	Word-by-word comparison on failure	Instead of just showing failure, the code attempts a basic word-level comparison to help the user understand *where* they made a mistake, enhancing usability.

9. Implementation Details

The project is implemented in two main functions:

1. **`calculate_wpm(time_taken, textlength)`:**
    - Takes the total time elapsed (in seconds) and the length of the text (in characters).
    - Calculates words typed: `textlength / 5`.
    - Converts time from seconds to minutes: `time_taken / 60`.
    - WPM is calculated and returned, rounded to the nearest integer. Includes a zero-division guard.
2. **`typing_test()`:**
    - Initializes a list of target sentences.
    - Uses `random.choice()` to select the test phrase.
    - Measures time using `time.time()`.
    - Uses `if/else` branching for result determination.
    - The mismatch handling logic iterates through the split words of both strings to provide targeted feedback on errors.

10. Screenshots / Results



```python
def typing_test():  #main function to run the typing test
        print("CONGRATULATIONS! Your answer is correct.")
        print(f"Your typing speed is: {wpm} words per minute (WPM)")
        print(f"Time taken: {total_time:.1f} seconds")
        print("#" * 50)
    else:
        print("\n" + "!" * 50)
        print("SORRY! Your answer is incorrect.")  #notify user of incorrect input
        print("Please try again for accurate WPM.")
        print(f"Time taken: {total_time:.1f} seconds")
        print(f"Provided Entry:{targettext}")
        print(f"Your Entry:{u_input}")   #show user input vs target text
        print("\nDifferences:")
        targetwords=targettext.strip().split()  #split target text into words
        u_words=u_input.strip().split()  #split user input into words
        for i in range(len(targetwords)):
            if i < len(u_words):
                if targetwords[i] != u_words[i]:
                    print(f"Mismatch at character {i+1}: expected '{targetwords[i]}', got '{u_words[i]}'")
            else:
```

```
*******************
TYPING SPEED TEST
**********************************************************************************************************************************
*******************
Type the following phrase exactly as it appears:
---------------------------------------------------------------------------


>>>>> The chocolate chip cookies smelled so good that I ate one without asking.<<<<


---------------------------------------------------------------------------
Type the above phrase here : The chocolate chip cookies smelled so good that I ate one without asking.

**************************************************
CONGRATULATIONS! Your answer is correct.
Your typing speed is: 43 words per minute (WPM)
Time taken: 20.4 seconds
##################################################
PS D:\New folder\cse_vityrthi>
```

11. Testing Approach

The testing approach utilized both unit and functional testing:

Test Type	Test Case	Expected Result
Unit Test (`calculate_wpm`)	Time: 10s, Length: 50 characters (10 words)	WPM = (10 words / (10/60) min) = 60 WPM
Functional Test (Correct Input)	User types phrase correctly	Output: "CONGRATULATIONS!", displays calculated WPM.
Functional Test (Incorrect Input)	User omits one word	Output: "SORRY!", displays the word mismatch, e.g., "expected 'over', got 'the'".
Functional Test (Whitespace)	User adds extra space at end	The `.strip()` function should ensure the input is still considered correct.

12. Challenges Faced

The project was intentionally kept simple, relying only on standard library features. The main subtle challenge involved ensuring robust input comparison:

- **Whitespace Handling:** Ensuring that minor variations in leading/trailing whitespace do not immediately cause a test failure, which was solved by applying the `.strip()` method to both the target text and the user input.

13. Learnings & Key Takeaways

This project reinforced understanding of:

- **Time Measurement in Python:** Practical application of `time.time()` for measuring elapsed duration.
- **String Manipulation:** Effective use of `str.split()`, `str.strip()`, and string indexing for data comparison and processing.
- **Standard WPM Definition:** Implementation of the industry-standard "5-character-per-word" rule for speed calculation.

14. Future Enhancements

Potential improvements for future development include:

1. **High Score Persistence:** Implement file I/O or a lightweight database (e.g., SQLite) to store user WPM records.
2. **Error-Adjusted WPM:** Calculate a more accurate "Net WPM" that penalizes based on the number of errors or incorrect characters.
3. **GUI Interface:** Develop a graphical user interface (using libraries like Tkinter or PyQt) to replace the basic console for a more user-friendly experience.
4. **Multi-Word Text Blocks:** Allow the test to use entire paragraphs instead of just single sentences.

15. References

- Python Standard Library Documentation (for `time` and `random` modules).