

Typing Tester

Team Candidates Number: 674

Applied Computer Information Technology, Oslo Metropolitan University

ACIT4420: Problem-Solving with Scripting

Report Submission Due Date: 07.12.2023

Report Title

1 Table of Contents

Executive Summary 5

1 Introduction..... 1

 1.1 Background 1

 1.2 Objectives 1

 1.3 Significance..... 1

 1.4 Scope 1

2 Literature Review 1

3 System Design and Architecture 3

 3.1 Application Framework and Design Principles..... 3

 3.1.1 Modularity: 3

 3.1.2 User-Centric Design: 3

 3.1.3 Performance Efficiency: 3

 3.2 Software Architecture..... 3

 3.2.1 Front-End (GUI): 4

 3.2.2 Back-End Logic: 4

 3.2.3 Data Storage: 4

 3.3 Key Components and Their Interactions 4

 3.3.1 Text Management: 4

 3.3.2 Performance Evaluation: 4

 3.3.3 Data Handling: 4

 3.3.4 Graphical Data Representation: 4

 3.4 Security and Data Integrity 4

 3.4.1 Data Validation: 4

 3.4.2 Error Handling: 4

4 Features and Functionalities 5

 4.1 User Interface and Experience 5

 4.1.1 Animated Text Label: 5

 4.1.2 Customizable Settings: 5

 4.1.3 Real-Time Error Highlighting: Error! Bookmark not defined.

 4.1.4 Voice Feedback for Typed Letters: 6

 4.2 Performance Tracking and Feedback 6

4.2.1	Instant Performance Metrics:	6
4.2.2	Historical Data Tracking:	6
4.2.3	Graphical Data Representation:	7
4.3	Customization and Flexibility	7
4.3.1	Theme Selection:	7
4.3.2	Font Customization:	7
4.3.3	Layout Adjustments:	8
4.3.4	Practice Mode:	8
4.3.5	Session Time Limits:	8
4.4	Data Management and Security	9
4.4.1	User Registration and Login System:	9
4.4.2	Input Validation for Login:	10
4.4.3	Performance Data Saving and Loading:	10
4.4.4	Secure Data Handling:	10
4.5	Community and Competition	11
4.5.1	Points and Level System:	11
4.5.2	Leaderboard Competitive Features:	11
5	Implementation Details	13
5.1	Programming Language and Libraries	13
5.1.1	Python:	13
5.2	Core Functionality Implementation	15
5.2.1	Text Loading and Randomization:	15
5.2.2	Accuracy Calculation:	15
5.2.3	Typing Speed Calculation:	15
5.3	GUI Design and Interaction	15
5.3.1	Layout Management:	15
5.3.2	Event Handling:	15
5.3.3	Error Highlighting:	15
5.4	Data Visualization	15
5.4.1	Bar Chart for Typing Speeds:	15
5.5	Practice Mode Feature	15
5.5.1	Toggle Functionality:	15
5.6	Code Optimization and Testing	16
5.6.1	Performance Optimization:	16
5.6.2	Testing:	16
6	Testing and Examples	17

6.1	Testing Methodology	17
6.1.1	Unit Testing:	17
6.1.2	Integration Testing:	17
6.1.3	User Acceptance Testing (UAT):	17
6.1.4	Performance Testing:	17
6.2	Test Cases and Results	17
6.2.1	Case 1: Beginner Level Typist:	17
6.2.2	Case 2: Average Typist:	17
6.2.3	Case 3: Advanced Typist:	18
6.2.4	Case 4: Long Text Input:	18
6.3	Examples of Application Use	18
6.3.1	Example 1: Learning Typing Skills:	18
6.3.2	Example 2: Average Skill Enhancement:	20
6.3.3	Example 3: Professional Skill Enhancement:	21
6.4	Challenges and Resolutions	23
6.4.1	Performance Optimization for Voice Feedback:	23
6.4.2	Elimination of Redundant Voice Feedback:	23
6.4.3	GUI Responsiveness:	23
6.4.4	Data Integrity:	23
6.4.5	User Experience:	23
6.5	Documentation and Code Comments.....	23
7	Analysis and Results.....	24
7.1	Application Performance	27
7.2	User Feedback and Improvements.....	27
7.3	Educational Impact	27
7.4	Future Enhancements	27
8	Discussion and Recommendations	28
9	Conclusion.....	29
9.1	Final Thoughts.....	29
10	References	30
11	Appendices	33
11.1	Appendix A	33
11.2	Appendix B	34

List of Figures

Figure 1: Animation of Texts	5
Figure 2: Highlighting of Typo-graphical Errors in Real-Time	5
Figure 3: Instant Performance Metrics	6
Figure 4: Tracking of Historical Data of a user	6
Figure 5: Selection of Theme	7
Figure 6: Customization of Font.....	8
Figure 7: Practice Mode - ON/OFF	8
Figure 8: Session Time Limit for Typing Practice	9
Figure 9: User Registration and Login System	9
Figure 10: Validation of Input for Login	10
Figure 11: Secure Data Handling.....	10
Figure 12: Points and Level System	11
Figure 13: Leaderboard Competitive Features	11
Figure 14: Example 1a: Learning Typing Skills Session One	18
Figure 15: Example 1b: Learning Typing Skills Session Two	19
Figure 16: Example 1c: Learning Typing Skills Session Three	19
Figure 17: Example 2a: Average Skill Enhancement Session One	20
Figure 18: Example 2b: Average Skill Enhancement Session Two	20
Figure 19: Example 2c: Average Skill Enhancement Session Three	21
Figure 20: Example 3a: Professional Skill Enhancement Session One.....	21
Figure 21: Example 3b: Professional Skill Enhancement Session Two	22
Figure 22: Example 3c: Professional Skill Enhancement Session Three	22
Figure 23: Example 1: Learning Typing Skills Results Analysis.....	24
Figure 24: Example 2: Average Skill Enhancement Results Analysis	25
Figure 25: Example 3: Professional Skill Enhancement Results Analysis.....	26

Executive Summary

The Typing Tester Application is a key tool for improving digital skills, especially in typing. This app is a fun and interactive place where people can get better at typing fast and accurately. It has lots of different texts to type, which is great for people at all skill levels, and it gives feedback right away to help users improve. One of the best things about the app is that it keeps track of how well you're doing, like how accurate you are and how many words you can type in a minute. This helps you see where you're improving and still need to work.

This app stands out because it's made for the user. It uses game-like elements, a simple countdown timer, speaking of words and letter dictating, highlighting typographical errors, moving text, and an easy-to-understand layout. This makes learning fun and a bit of a challenge. The app uses Tkinter for its graphics, which makes it easy to use, and Matplotlib to show how you're doing over time, making the whole experience better. This summary talks about the main features of the Typing Tester Application and how it can be a great tool for personal and educational growth in typing.

Keywords: *Typing Test, User Interface (UI) Design, Accuracy Measurement, Words Per Minute (WPM), Performance Feedback, Practice Mode, Time Management, Interactive Learning, Python Programming, Educational Tool.*

1 Introduction

1.1 Background

In today's world, being good at typing is more than just a skill – it's a must-have. The Typing Tester Application started as a simple way to help people type better. But as time went on, it grew a lot. Now, it has many cool features and is easy to use, making learning to type better and more fun. This change shows how important it is to be good at using digital tools and why we need good ways to learn these skills.

1.2 Objectives

The main goal of this Typing Tester Application is to be a fun and complete way for people to get faster and more accurate at typing. It wants to:

- Give users different kinds of typing tests with lots of text options.
- Show how well you're doing right away.
- Let users change their typing practice to fit what they need.
- Have a practice mode so you can learn without stress.
- Use cool tools to show users how they're getting better over time.

1.3 Significance

What makes this Typing Tester Application special is how it teaches typing in a fun and useful way. It turns the usual typing practice into something fun and rewarding. The app is great because it shows you how you're doing right away. This helps users take control of their learning, making them better typists and giving them more confidence and motivation to keep getting better.

1.4 Scope

This Typing Tester Application covers a few important areas:

- Making typing fun: It has cool features like moving text and a nice design.
- Keeping track of how you're doing: It saves and loads your typing data so you can see your progress.
- Letting you choose how you practice: You can set up your typing practice to match your skill level and goals.
- Building a community: It encourages users to connect and compete in a friendly way.

Right now, the app is a great tool for all kinds of typists, from beginners to experts. It's good for both people learning on their own and for use in schools or other places.

Literature Review

This section of my report critically examines and synthesizes relevant existing literature, research studies, and theoretical frameworks related to typing proficiency, typing training

software, and user interface design for educational tools. The section aims to highlight the current state of knowledge in these areas and identify gaps that this project addresses.

In a notable study, [1] the influence of word prediction software on spelling accuracy and typing speed was investigated. Focusing on children with spelling difficulties, the study involved 80 Grade 4 – 6 students from a special needs school. Participants were tasked with entering words using an on-screen keyboard, both with and without word-prediction software. The research found that while word prediction improved spelling accuracy, it also led to increased time consumption and a tendency to use word approximations. Interestingly, no significant relationship was found between the children's existing spelling knowledge and the efficacy of word prediction. This study highlights the potential benefits and drawbacks of using assistive technology in typing education, especially for individuals with specific learning challenges.

[2]research offers a comprehensive analysis of the effectiveness of rapid typing software and modules in improving typing skills. The study was conducted with a focus on students in a vocational high school (SMK N 7 Yogyakarta), where mastering typing skills is a crucial part of the curriculum. The research involved 30 students and employed methods like text typing skill documentation and Multivariate Analysis of Variance (MANOVA) for data analysis. The findings revealed that while the rapid typing software significantly improved typing speed, it did not have a marked impact on typing accuracy. This underscores the software's effectiveness in enhancing certain aspects of typing proficiency, particularly speed, but also highlights the need for a more holistic approach that addresses accuracy.

The study [3], [4] investigates the intricate balance between speed and accuracy in skilled typewriting, a crucial aspect of typing training programs. Their research highlights the hierarchical nature of typing control systems, consisting of two primary loops: the outer loop, which focuses on word generation, and the inner loop, responsible for the sequential activation and execution of keystrokes. Their key findings include:

- i. **Speed-Accuracy Trade-off:** Typists can modulate their speed for better accuracy, but there's a speed limit due to inherent process limitations.
- ii. **Inner-Loop Focus:** Most adjustments in typing speed and accuracy are attributed to the inner loop, highlighting its significance in typing performance.
- iii. **Outer-Loop's Role:** The outer loop, though less influential in speed adjustments, significantly impacts error rates, emphasizing the need for balanced training that addresses both loops.

[5] study (1954) at Leon High School, Tallahassee, Florida, offers insightful perspectives on the effectiveness of typing training programs, particularly in identifying and addressing common errors. Her research, conducted with a second-year typewriting class, aimed to analyze the types of typewriting errors prevalent among students and to understand the underlying causes of these errors. This analysis is crucial in tailoring typing programs to effectively address individual and collective challenges faced by students. Barrineau's findings contribute to a deeper understanding of how targeted interventions in typing education can enhance overall proficiency and accuracy, a key aspect often emphasized in modern typing software.

[6] Mavis Beacon Teaches Typing for Teachers exemplifies the impact of user-friendly design in educational software, particularly in typing tutors. This program stands out for its personalized learning approach, adapting to each student's progress and offering tailored exercises. Such customization is crucial for accommodating diverse learning styles and paces. A key feature of Mavis Beacon is its real-time progress tracking, providing immediate feedback to learners and valuable insights for educators. This functionality enhances the assessment process, making it more efficient and informative. The software's flexibility in lesson customization and goal setting aligns with the current educational emphasis on personalized learning outcomes. Interactive elements like typing games are integrated to maintain engagement, making learning both fun and effective.

While the solutions mentioned above have significantly contributed to typing education and assessment, there remains room for innovation. Some platforms might excel in engagement but lack detailed performance analytics. Others might offer comprehensive training but fall short in user experience. Recognizing these gaps and the evolving needs of digital natives, the "Typing Tester" project seeks to carve its niche, blending the best of assessment, feedback, and user engagement.

System Design and Architecture

1.5 Application Framework and Design Principles

This Typing Tester Application is built using the Python programming language, leveraging the Tkinter library for its graphical user interface (GUI). This choice ensures cross-platform compatibility and ease of use. The application adheres to the following design principles:

- 1.5.1 **Modularity:** The code is structured into distinct functions and classes, facilitating ease of maintenance and scalability. This main code file is structured into global functions, classes, and local functions or methods.
- 1.5.2 **User-Centric Design:** The GUI is designed with a focus on simplicity and intuitiveness, ensuring a seamless user experience.
- 1.5.3 **Performance Efficiency:** Efficient algorithms and data structures are used to ensure quick response times and minimal resource usage.

1.6 Software Architecture

The application follows a straightforward architecture:

- 1.6.1 **Front-End (GUI):** This was developed using Tkinter, it provides an interactive interface for users to engage with the application. It includes text display areas, input fields, buttons, progress bar, level-up points, and graphical elements for displaying performance data.
- 1.6.2 **Back-End Logic:** This was written in Python, and it handles the core functionalities such as text processing, accuracy calculation, performance tracking, and data management.
- 1.6.3 **Data Storage:** Over 1000 text sentences in TXT format and user performance data are stored in CSV format respectively, allowing for easy access and manipulation. This includes user IDs, session dates, accuracy, and WPM. While the text sentences are derived from different fields of subjects such as Biblical texts, technological texts, agricultural texts, business and marketing fields[7]–[16].

1.7 Key Components and Their Interactions

- 1.7.1 **Text Management:** The application loads text examples from a file (sentences.txt), where it randomly selects and presents them to the user for typing practice.
- 1.7.2 **Performance Evaluation:** After each session, the application calculates typing speed (WPM) and accuracy, providing immediate feedback to the user.
- 1.7.3 **Data Handling:** Performance data is saved to and loaded from a CSV file (user_performance-data.csv), ensuring the persistence of user progress.
- 1.7.4 **Graphical Data Representation:** This application utilizes Matplotlib for generating graphs, offering users a visual representation of their progress.

1.8 Security and Data Integrity

- 1.8.1 **Data Validation:** The application ensures that user inputs are validated before processing to prevent errors and maintain data integrity.
- 1.8.2 **Error Handling:** Robust error handling mechanisms are in place to manage file access and data storage operations, ensuring the application's stability.

2 Features and Functionalities

2.1 User Interface and Experience

The Typing Tester Application is meticulously designed to prioritize user experience, ensuring that both new and experienced users find the interface engaging and easy to navigate. Key elements of the user interface and experience include: Figure 1

- 2.1.1 **Animated Text Label:** This feature enhances user engagement with a welcoming, dynamic display.

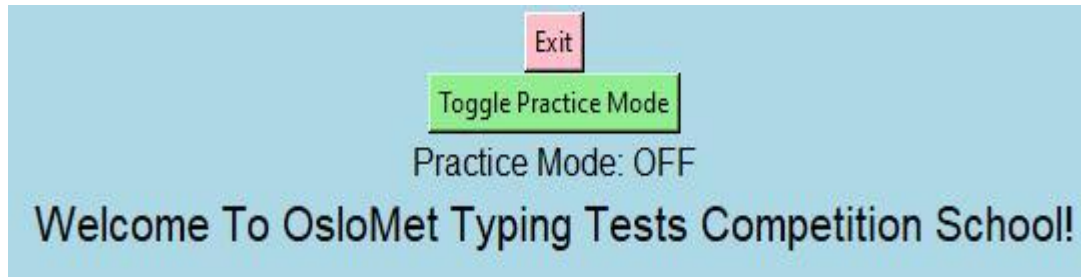


Figure 1: Animation of Texts

- 2.1.2 **Customizable Settings:** Understanding that users have varying skill levels and learning preferences, the application offers customizable settings. This includes the ability to adjust the number of examples or typing tests in each session. Such flexibility accommodates beginners who might prefer shorter sessions and advanced users who seek more extensive practice. This customization extends to other aspects like text size and color, ensuring comfort and accessibility for all users.
- 2.1.3 **Real-Time Error Highlighting:** One of the application's standout features is its ability to highlight errors in real-time as the user types. This immediate feedback is crucial for learning and correction, allowing users to quickly identify and understand their mistakes. The visual distinction of errors helps in reinforcing correct typing habits, significantly enhancing the learning curve.

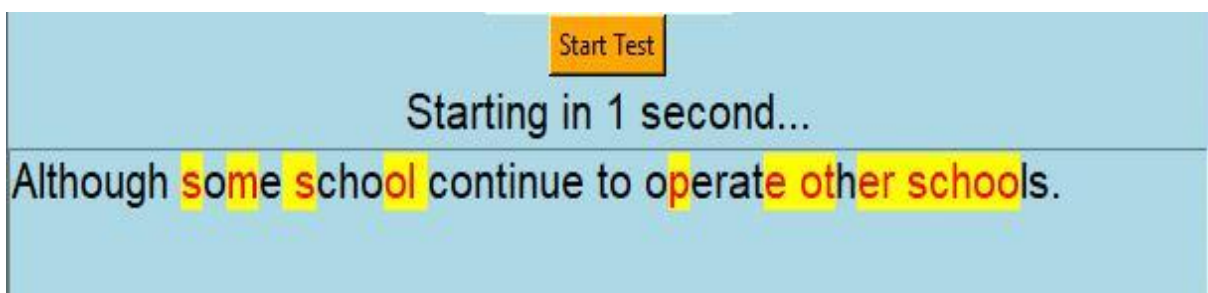


Figure 2: Highlighting of Typo-graphical Errors in Real-Time

2.1.4 Voice Feedback for Typed Letters: This feature enhances the learning experience by providing real-time auditory feedback. As the user types each letter of the displayed phrase or sentence, the application pronounces the letter. This immediate feedback helps users in associating the keyboard layout with the corresponding sounds, which can be particularly beneficial for beginners or those looking to improve their typing accuracy and speed.

2.2 Performance Tracking and Feedback

The application is not just a tool for practicing typing but also a comprehensive system for tracking and improving typing skills. It achieves this through:

2.2.1 Instant Performance Metrics: Immediately after each typing session, the application provides users with detailed feedback on their performance. This includes metrics like typing accuracy and words per minute (WPM), crucial indicators of typing proficiency. Such instant feedback allows users to gauge their performance in real time, making the learning experience more tangible and rewarding.

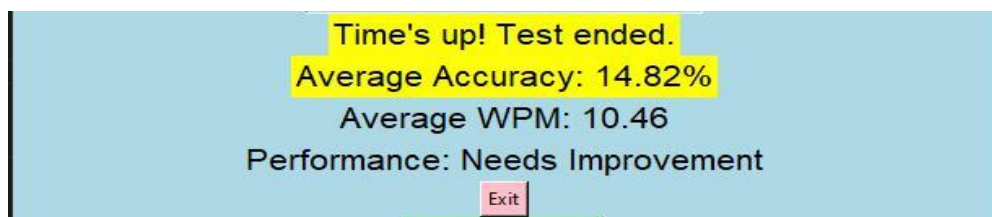


Figure 3: Instant Performance Metrics

2.2.2 Historical Data Tracking: Beyond immediate feedback, the application also offers historical data tracking. This feature allows users to view and analyze their performance over time. By having access to their progress history, users can identify patterns, celebrate improvements, and pinpoint areas needing more practice. This longitudinal view of performance is instrumental in setting and achieving personal typing goals.

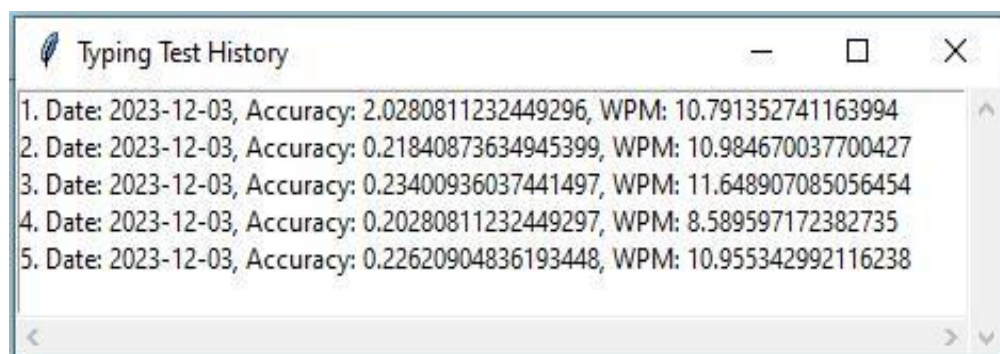


Figure 4: Tracking of Historical Data of a user

2.2.3 Graphical Data Representation: To analyze progress both intuitive and insightful, the application incorporates graphical representations of data. Users can view their improvements in typing speed and accuracy through easy-to-understand graphs and charts. These visual tools transform raw data into actionable insights, making it easier for users to comprehend their progress and stay motivated. The graphical representation is particularly effective in showcasing long-term progress and helping users set realistic and achievable goals based on their performance trends.

2.3 Customization and Flexibility

The Typing Tester Application is designed with a deep understanding of diverse user needs, offering customization and flexibility that allows users to personalize the visual aspects of the application according to their preferences. Customization can greatly enhance the user experience by making the application more visually appealing and comfortable to use, especially during extended typing sessions. Key aspects of this feature include:

2.3.1 Theme Selection: Users can choose from a variety of themes, which may include different color schemes, font styles, and background images. This allows users to select a visual presentation that is most pleasing to their eyes or that best fits the lighting conditions of their environment.

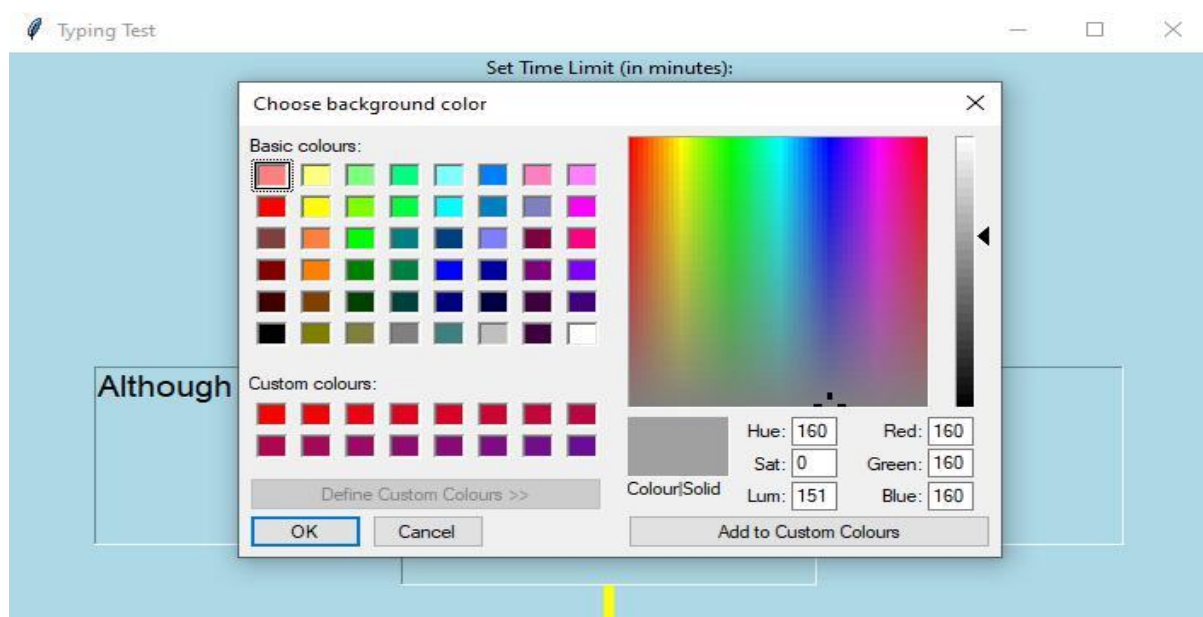


Figure 5: Selection of Theme

2.3.2 Font Customization: The application allows users to change the font style and size of the text displayed. This is particularly useful for users who may have visual impairments or preferences for certain types of fonts for better readability.

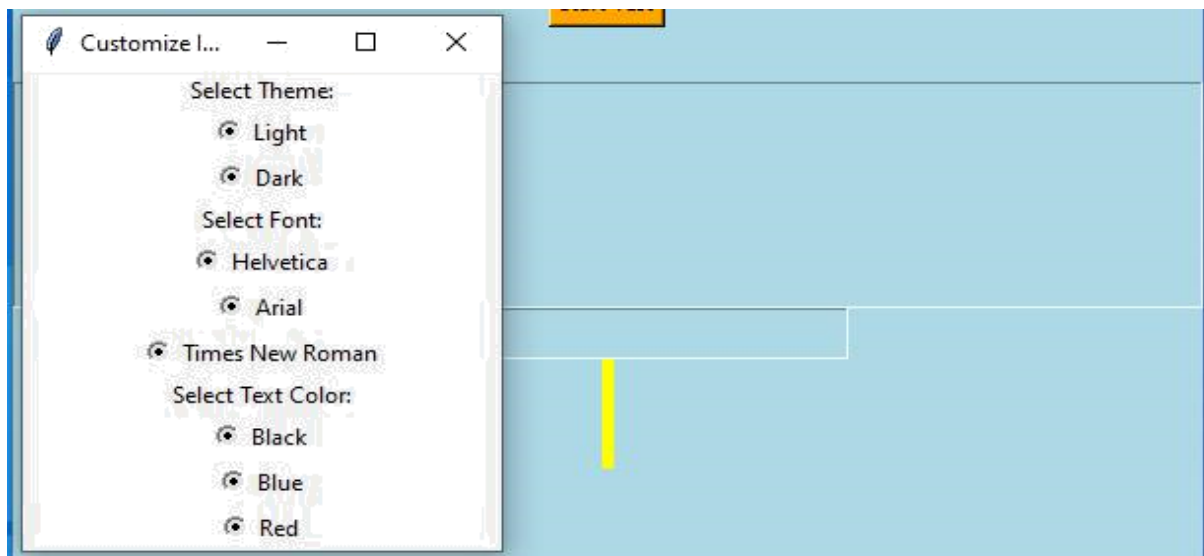


Figure 6: Customization of Font

2.3.3 Layout Adjustments: Users can adjust the layout of the application, such as the size and positioning of the typing area, scoreboard, and other elements. This flexibility ensures that the application can adapt to different screen sizes and user preferences, making it more accessible and user-friendly.

2.3.4 Practice Mode: This mode is a cornerstone of the application's adaptability. It provides a relaxed, stress-free environment ideal for beginners or those looking to hone their skills without the added pressure of competing against the clock. In this mode, users can focus solely on accuracy and technique, building confidence before moving on to more challenging timed tests.

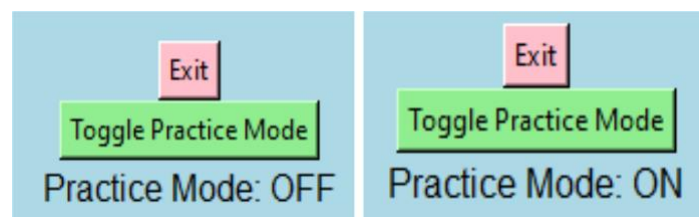


Figure 7: Practice Mode - ON/OFF

2.3.5 Session Time Limits: Recognizing that users have different availability and attention spans; the application allows the setting of time limits for typing sessions in minutes but automatically converted to seconds upon the starting of typing. This feature adds significant flexibility to practice routines, accommodating those who prefer short, focused sessions as well as users who wish for longer, more intensive practice periods. This customization ensures that the application fits seamlessly into various daily schedules, enhancing its usability.

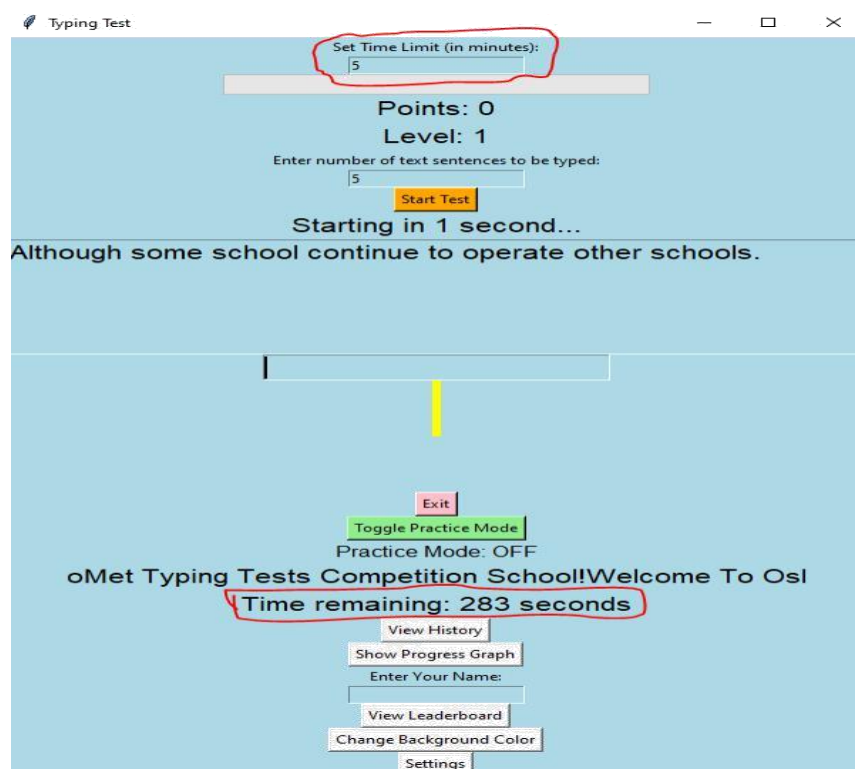


Figure 8: Session Time Limit for Typing Practice

2.4 Data Management and Security

In an era where data security is paramount, the Typing Tester Application places a high priority on robust data management and security:

2.4.1 User Registration and Login System: Before user can gain access to the application, such a user must first register. The application includes a user login system, requiring users to enter an existing username and a unique ID before starting the typing test. This feature adds a personalized touch to the application, allowing for a more tailored user experience. It also paves the way for future enhancements, such as tracking individual progress over time or customizing tests based on user performance.

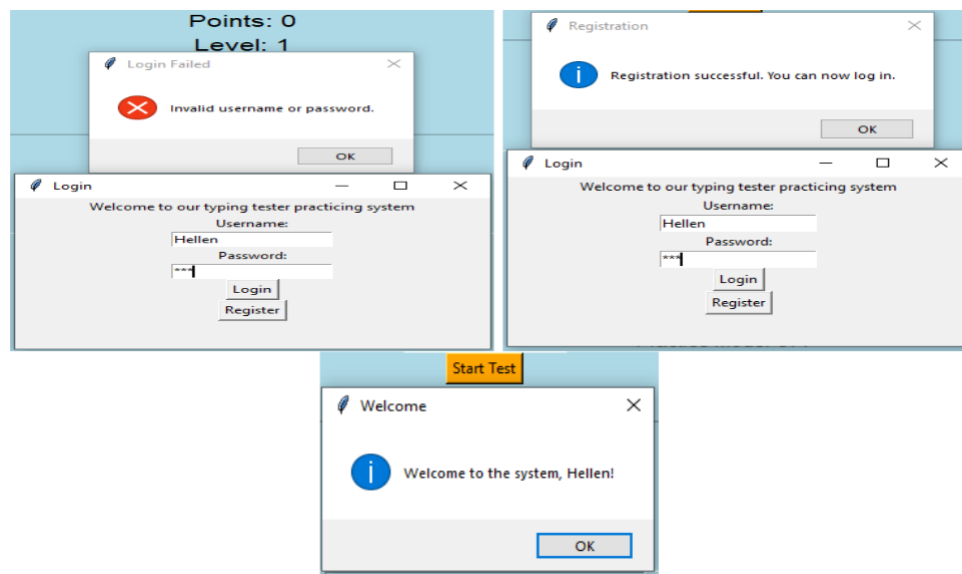


Figure 9: User Registration and Login System

2.4.2 Input Validation for Login: To ensure data integrity and a smoother user experience, input validation checks have been implemented in the login system. This feature verifies that the username and user ID meet certain criteria (such as non-emptiness, length, or format) before allowing access to the typing test. This validation helps prevent errors and potential misuse, ensuring a more robust and user-friendly application.



Figure 10: Validation of Input for Login

2.4.3 Performance Data Saving and Loading: The application ensures that user performance data is not only saved but also easily retrievable. This functionality means that users can pick up right where they left off, with no loss of progress or data. It allows for a continuous learning experience, where each session's data contributes to a comprehensive view of the user's improvement over time.

2.4.4 Secure Data Handling: The application employs advanced measures to ensure the integrity and privacy of user data. This includes secure data storage practices and encryption protocols to protect sensitive information (e.g. passwords). Users can trust that their data is handled with the utmost care, safeguarding their personal information while they focus on improving their typing skills.



Figure 11: Secure Data Handling

2.5 Community and Competition

The application not only focuses on individual progress but also fosters a sense of community and healthy competition:

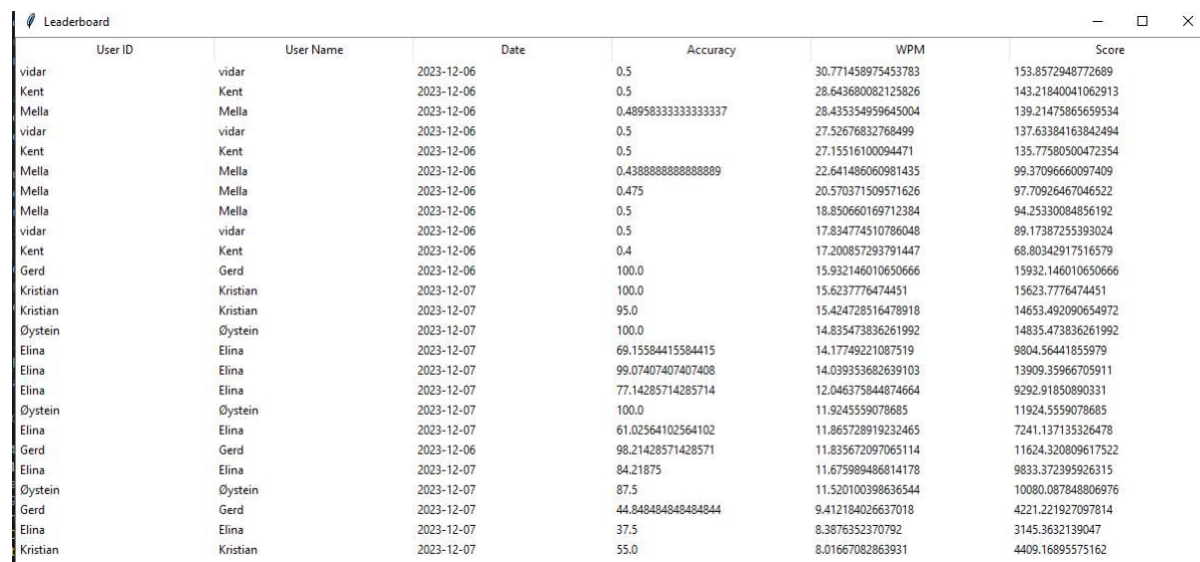
2.5.1 Points and Level System: As users engage with the application, they earn points and level up based on their performance. This gamified element adds an extra layer of motivation and enjoyment to the learning process. It encourages users to set goals and strive for improvement, turning the often-monotonous task of typing practice into an engaging and rewarding experience. Figure 12



The screenshot shows a window titled "Typing Test". Inside, there is a "Set Time Limit (in minutes):" field with a text input box. Below this, a large light blue box displays "Points: 0" and "Level: 1". Underneath, there is a label "Enter number of text sentences to be typed:" followed by another text input box. At the bottom of this box is a yellow "Start Test" button.

Figure 12: Points and Level System

2.5.2 Leaderboard Competitive Features: Looking toward future updates, the application plans to introduce features that allow users to compare their performance with others. This will encourage a competitive yet supportive environment, where users can challenge themselves against peers, share tips, and celebrate each other's progress. Such community-oriented features are aimed at enhancing user engagement and fostering a supportive network of learners.



User ID	User Name	Date	Accuracy	WPM	Score
vidar	vidar	2023-12-06	0.5	30.771458975453783	153.8572948772689
Kent	Kent	2023-12-06	0.5	28.643680082125826	143.21840041062913
Mella	Mella	2023-12-06	0.4895833333333333	28.435354959645004	139.21475865659534
vidar	vidar	2023-12-06	0.5	27.52676832768499	137.63384163842494
Kent	Kent	2023-12-06	0.5	27.15516100094471	135.77580500472354
Mella	Mella	2023-12-06	0.4388888888888889	22.641486060981435	99.3709660097409
Mella	Mella	2023-12-06	0.475	20.570371509571626	97.70926467046522
Mella	Mella	2023-12-06	0.5	18.850660169712384	94.25330084856192
vidar	vidar	2023-12-06	0.5	17.834774510786048	89.17387255393024
Kent	Kent	2023-12-06	0.4	17.200857293791447	68.80342917516579
Gerd	Gerd	2023-12-06	100.0	15.932146010650666	15932.146010650666
Kristian	Kristian	2023-12-07	100.0	15.6237776474451	15623.7776474451
Kristian	Kristian	2023-12-07	95.0	15.424728516478918	14653.492090654972
Oystein	Oystein	2023-12-07	100.0	14.835473836261992	14835.473836261992
Elina	Elina	2023-12-07	69.15584415584415	14.17749221087519	9804.56441855979
Elina	Elina	2023-12-07	99.07407407407408	14.039353682639103	13909.35966705911
Elina	Elina	2023-12-07	77.14285714285714	12.046375844874664	9292.91850890331
Oystein	Oystein	2023-12-07	100.0	11.9245559078685	11924.5559078685
Elina	Elina	2023-12-07	61.02564102564102	11.865728919232465	7241.137135326478
Gerd	Gerd	2023-12-06	98.21428571428571	11.835672097065114	11624.320809617522
Elina	Elina	2023-12-07	84.21875	11.675989486814178	9833.372395926315
Oystein	Oystein	2023-12-07	87.5	11.520100398636544	10080.087848806976
Gerd	Gerd	2023-12-07	44.848484848484844	9.412184026637018	4221.221927097814
Elina	Elina	2023-12-07	37.5	8.3876352370792	3145.3632139047
Kristian	Kristian	2023-12-07	55.0	8.01667082863931	4409.16895575162

Figure 13: Leaderboard Competitive Features

The *Leaderboard* feature in my Typing Test Application serves as a motivational and performance-tracking tool. It is designed to foster a competitive and engaging environment for users by displaying a ranked list of top performers based on their typing proficiency. This feature not only encourages users to improve their typing skills but also allows them to see where they stand in comparison to others.

3 Implementation Details

3.1 Programming Language and Libraries

3.1.1 Python: The choice of Python for this project was driven by its simplicity and the vast array of libraries it offers. Python's syntax is clear and readable, which is beneficial for educational tools where code maintenance and understanding are crucial.

3.1.1.1 Tkinter: *This library was selected for its integration with Python's standard library, making it a reliable choice for GUI development. We leveraged Tkinter's widgets like buttons, text fields, and labels to create an intuitive user interface. The decision to use Tkinter also stems from its cross-platform compatibility, ensuring the application runs smoothly on different operating systems[17]–[23][24]–[30].*

3.1.1.2 matplotlib: *The inclusion of Matplotlib serves the purpose of enhancing user engagement through visual feedback. The bar chart created using this library offers an immediate, visual representation of the user's performance, making the data more accessible and understandable[17]–[23][24]–[30].*

3.1.1.3 threading and Time: *These modules are crucial for managing the application's time-based features concurrently. The threading module allows the application to perform countdowns, speech synthesis, and track typing durations without freezing the GUI, ensuring a smooth user experience [17]–[23][24]–[30].*

3.1.1.4 csv: *This module is used for reading from and writing to CSV (Comma comma-separated values) files. The CSV is employed in this application for storing and retrieving user performance data and user credentials, allowing for easy data management and analysis[24]–[30].*

3.1.1.5 Os: *The os module provides a way of using operating system-dependent functionality. In the context of this application, it is used for file path manipulations or to interact with the underlying operating system for file operations[17]–[23][24]–[30].*

3.1.1.6 datetime and timedelta: *These are part of Python's datetime module and are used for handling and manipulating dates and times. In this application, they are used to record the time and date of each typing session and to handle time-based calculations like session durations[17]–[23][24]–[30].*

- 3.1.1.7 ***tkinter.messagebox and tkinter.ttk:*** These are submodules of Tkinter. The message box submodule is used for displaying pop-up messages to the user, such as error alerts or informational dialogs. The ttk submodule (themed Tkinter) provides access to Tkinter's themed widget set, which can be used to create a more modern and visually appealing GUI[17]–[23][24]–[30].
- 3.1.1.8 ***time:*** This module provides various time-related functions. In the application, it's used alongside threading for managing time-dependent features like timers and delays[17]–[23][24]–[30].
- 3.1.1.9 ***random:*** This module implements pseudo-random number generators for various distributions. The random is used in the application to randomly select text sentences for the typist to type, ensuring a varied and unpredictable typing experience [17]– [23][24]–[30].
- 3.1.1.10 ***matplotlib.pyplot:*** This is a submodule of Matplotlib, a comprehensive library for creating static, animated, and interactive visualizations in Python. In the application, pyplot is used to generate graphs and charts that visually represent the user's typing performance over time[17]–[23][24], [25], [27]–[30].
- 3.1.1.11 ***numpy (np):*** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It could be used in this application for efficient numerical computations, especially when dealing with large datasets or complex calculations[17]–[23][24], [25], [27]–[30].
- 3.1.1.12 ***Pytsx3:*** This is a text-to-speech conversion library in Python. It's used in the application to read aloud the text that the user needs to type, enhancing the interactivity of the application, especially for users with visual impairments[17]–[23][24], [25], [27]–[30].
- 3.1.1.13 ***queue, Queue:*** This module provides a queue implementation in Python. In the application, it's used for managing the speech queue in the text-to-speech feature, ensuring that text is spoken in the order it's received and without blocking the main application thread[17]–[23][24], [25], [27]–[30].
- 3.1.1.14 ***tkinter.colorchooser:*** This is another submodule of Tkinter, providing a dialog allowing the user to choose a color. In the application, it's used to let users customize the color scheme of the GUI, adding a level of personalization to the user experience[17]– [23][24], [25], [27]–[30].

3.2 Core Functionality Implementation

- 3.2.1 **Text Loading and Randomization:** This feature enhances the application's utility by providing a diverse range of typing text challenges. The ability to load different text sentences means the application can cater to various skill levels and interests.
- 3.2.2 **Accuracy Calculation:** The method of calculating accuracy is a key metric in typing tests. This function not only assesses the user's performance but also provides essential feedback, which is crucial for learning and improvement.
- 3.2.3 **Typing Speed Calculation:** The application's ability to calculate WPM aligns with industry standards for typing tests. This feature is vital for users to track their progress and set personal goals.

3.3 GUI Design and Interaction

- 3.3.1 **Layout Management:** The use of Tkinter's pack geometry manager was a strategic decision to balance simplicity and effectiveness in layout design. This choice is particularly beneficial for developers new to GUI programming.
- 3.3.2 **Event Handling:** The application's responsiveness and interactivity are achieved through event-driven programming. This approach ensures that the application reacts in real-time to user inputs, enhancing the overall user experience.
- 3.3.3 **Error Highlighting:** Immediate error feedback is essential in learning environments. This feature helps users quickly identify and correct their mistakes, which is a critical part of the learning process in typing.

3.4 Data Visualization

- 3.4.1 **Bar Chart for Typing Speeds:** The decision to include a bar chart was made to provide users with a clear and engaging way to view their progress. This visual representation helps in making the abstract data more tangible and understandable.

3.5 Practice Mode Feature

- 3.5.1 **Toggle Functionality:** The inclusion of a practice mode offers flexibility to the user, catering to both beginners and advanced users. This mode is especially useful for users who want to practice without the pressure of a timed environment. It could be turned On or Off by the users.

3.6 Code Optimization and Testing

3.6.1 **Performance Optimization:** The application was optimized for performance to handle multiple functionalities efficiently. This optimization ensures that the application remains responsive and user-friendly, even on lower-end devices.

3.6.2 **Testing:** The application underwent thorough testing, including unit tests for individual functions and comprehensive user acceptance testing. This testing ensures reliability and robustness, crucial for educational software.

4 Testing and Examples

4.1 Testing Methodology

- 4.1.1 **Unit Testing:** Each function, from text loading to accuracy calculation, was subjected to unit testing. This ensured that individual components of the application worked as expected. For instance, the accuracy calculation function was tested with predefined inputs to ensure it returned the correct accuracy percentage.
- 4.1.2 **Integration Testing:** After unit testing, integration testing was conducted to ensure that different parts of the application worked together seamlessly. This included testing the interaction between the GUI elements and the underlying logic, such as the response of the application to user inputs.
- 4.1.3 **User Acceptance Testing (UAT):** A group of users with varying typing skills was selected to test the application. Their feedback was crucial in understanding the user experience, including the application's usability, interface design, and overall functionality.
- 4.1.4 **Performance Testing:** Assessing the application's response time and resource usage under different scenarios.

4.2 Test Cases and Results

- 4.2.1 **Case 1: Beginner Level Typist:** A user with basic typing skills was asked to use the application. The focus was on the application's ability to handle slow typing speeds and frequent errors. The error highlighting feature and the adaptive difficulty level were particularly useful in this scenario.
- 4.2.2 **Case 2: Average Typist:** An average typist tested the application for its ability to accurately track high typing speeds and provide challenging text sentences. The application's performance in handling fast inputs and providing real-time feedback was noted.

4.2.3 **Case 3: Advanced Typist:** An experienced typist tested the application for its ability to accurately track high typing speeds and provide challenging text sentences. The application's performance in handling fast inputs and providing real-time feedback was noted.

4.2.4 **Case 4: Long Text Input:** The application was tested with a longer text to evaluate its performance under extended typing sessions. This test was crucial to assess the application's stability and memory management over longer usage periods.

4.3 Examples of Application Use

In each example case here, the participant typists have individually taken three different typing sessions.

4.3.1 **Example 1: Learning Typing Skills:** A beginner typist used the application for four days. Their progress in typing speed and accuracy was recorded, demonstrating the application's effectiveness as a learning tool. Figure 14

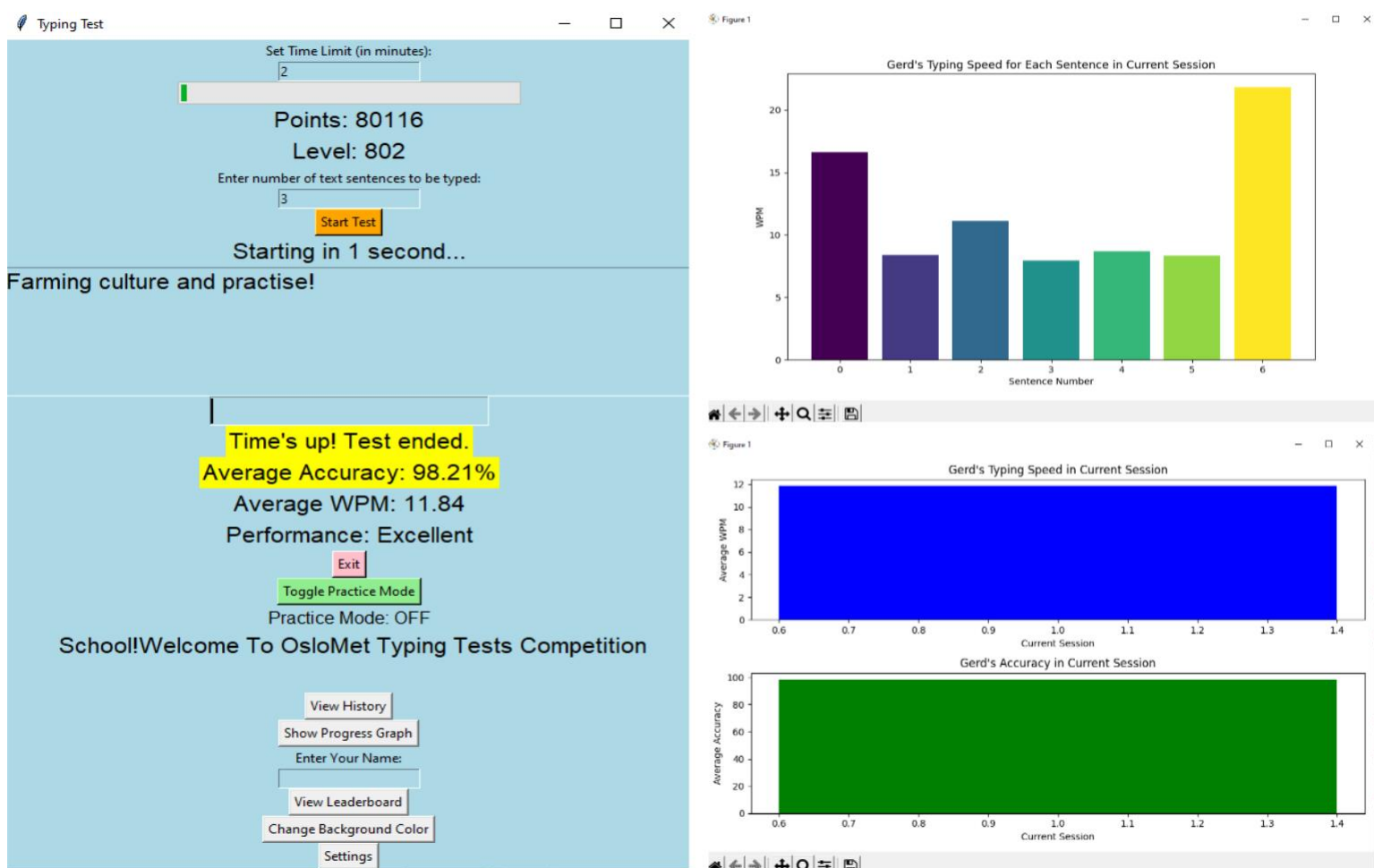


Figure 14: Example 1a: Learning Typing Skills Session One

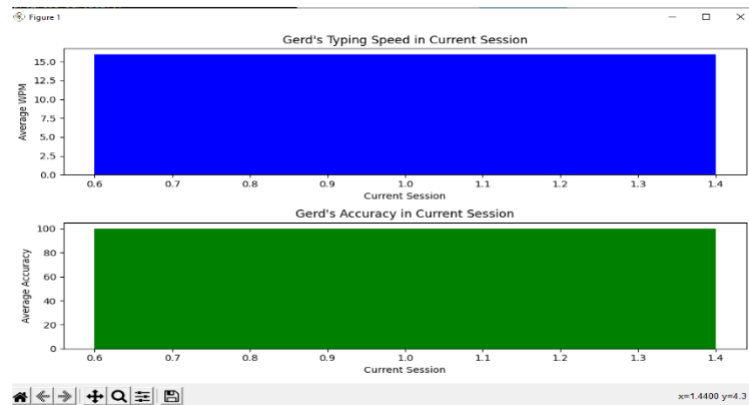
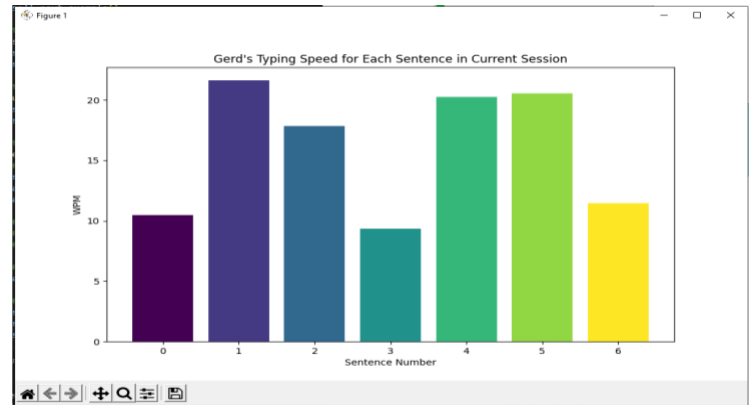
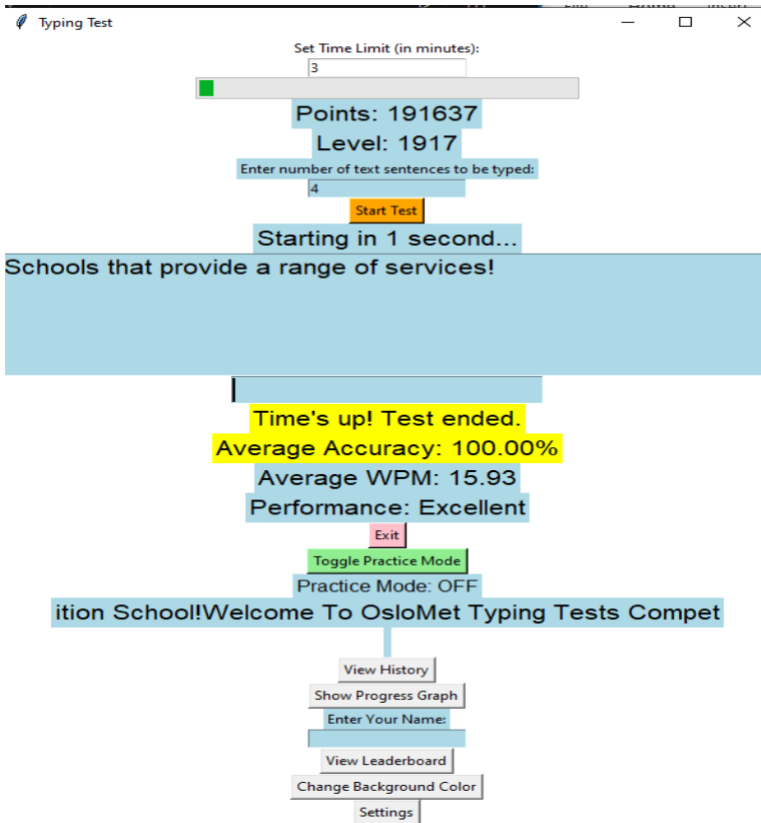


Figure 15: Example 1b: Learning Typing Skills Session Two

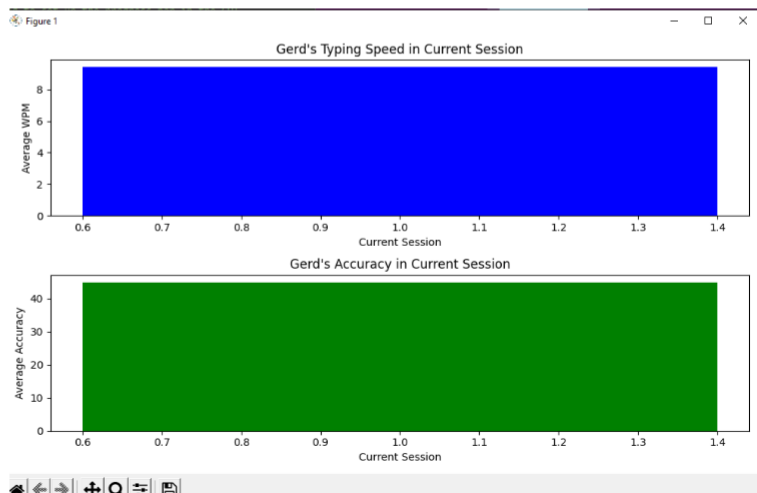
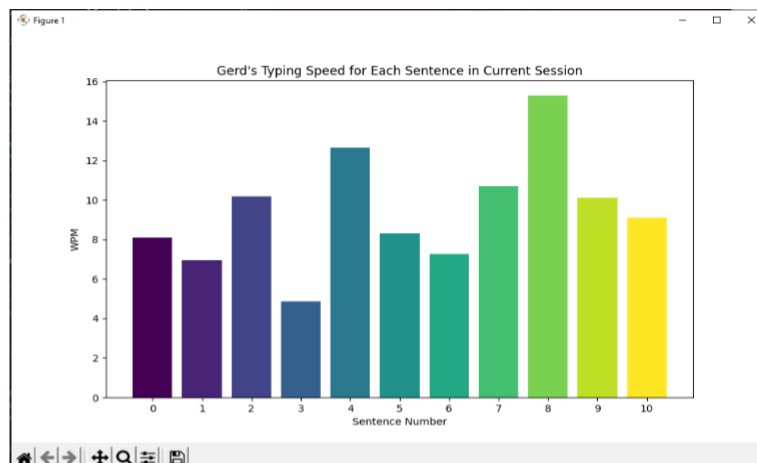
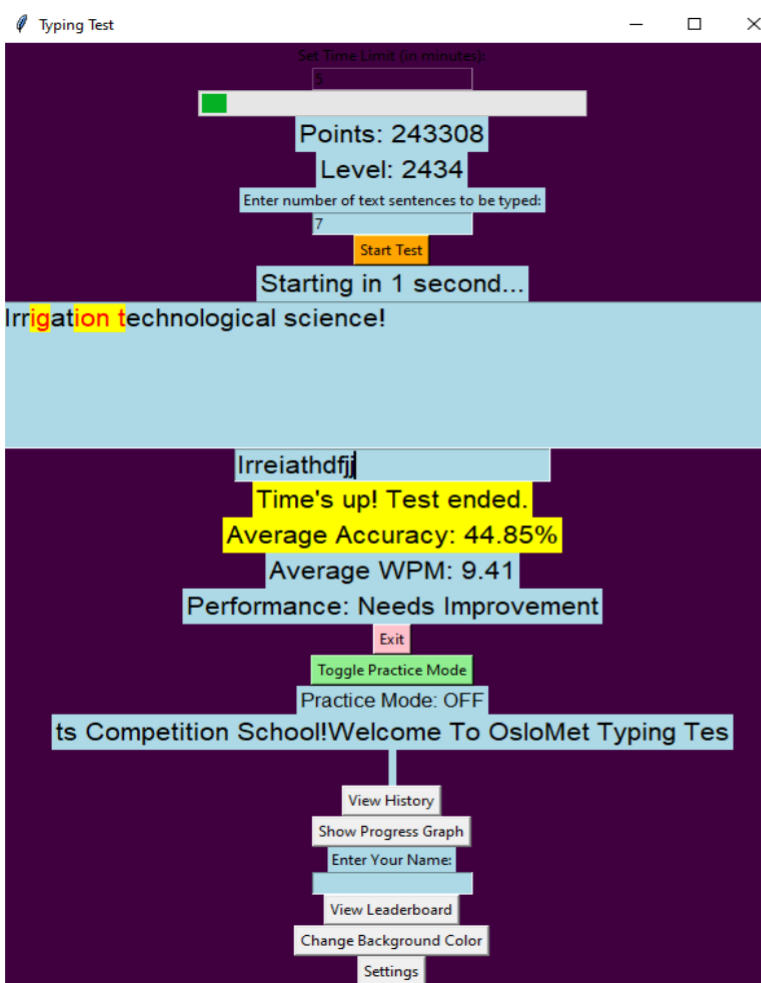


Figure 16: Example 1c: Learning Typing Skills Session Three

4.3.2 **Example 2: Average Skill Enhancement:** An average typist used the application for four days. Their progress in typing speed and accuracy was recorded, demonstrating the application's effectiveness as a learning tool. Figure 17

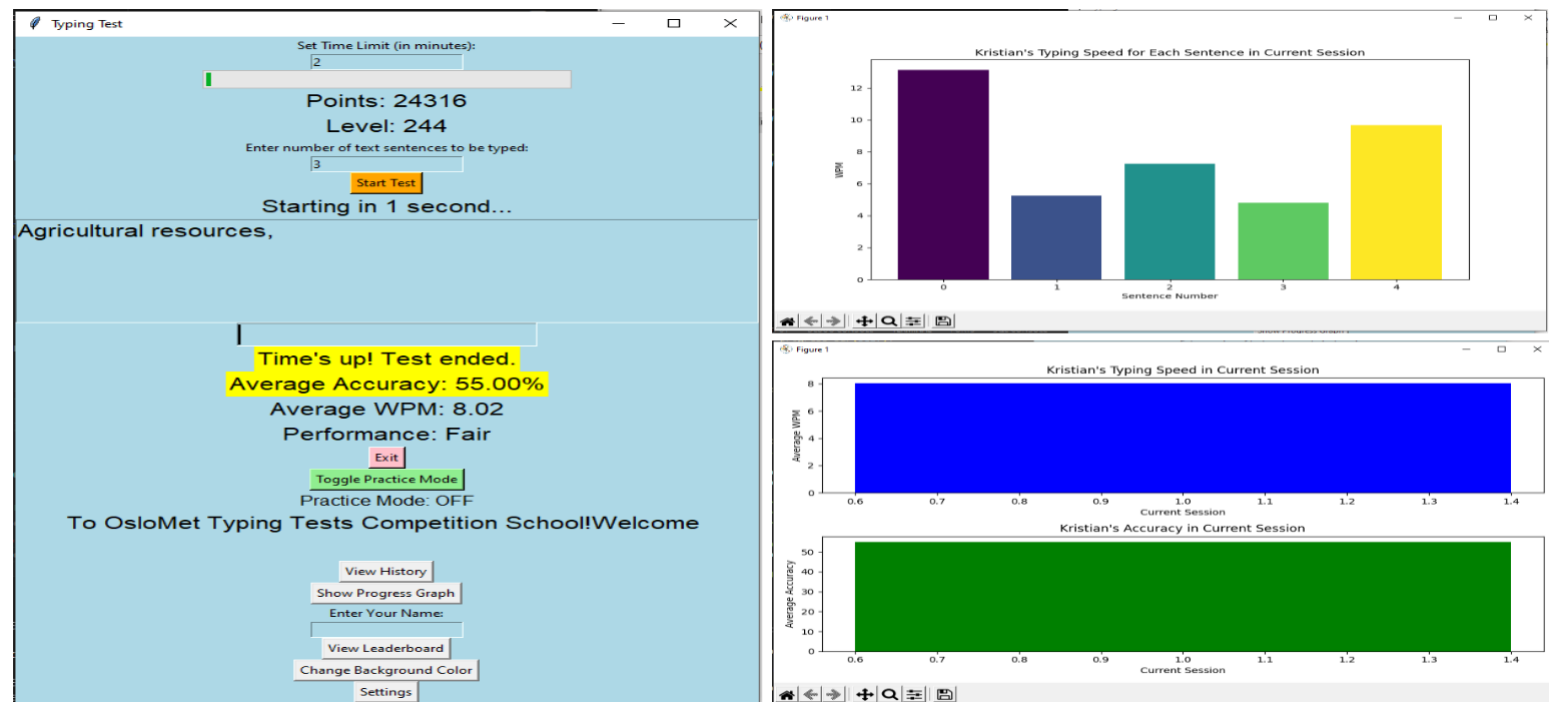


Figure 17: Example 2a: Average Skill Enhancement Session One

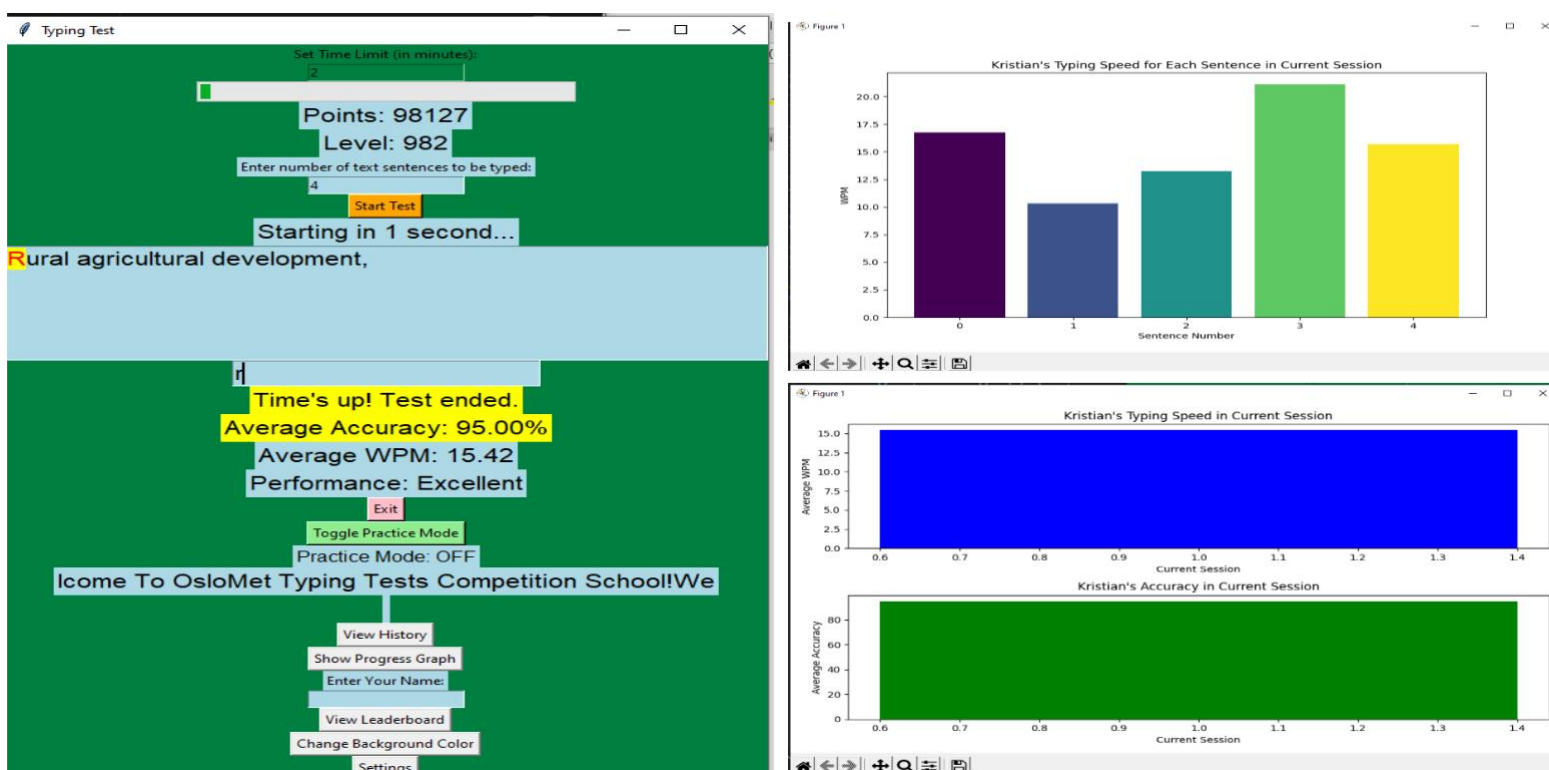


Figure 18: Example 2b: Average Skill Enhancement Session Two

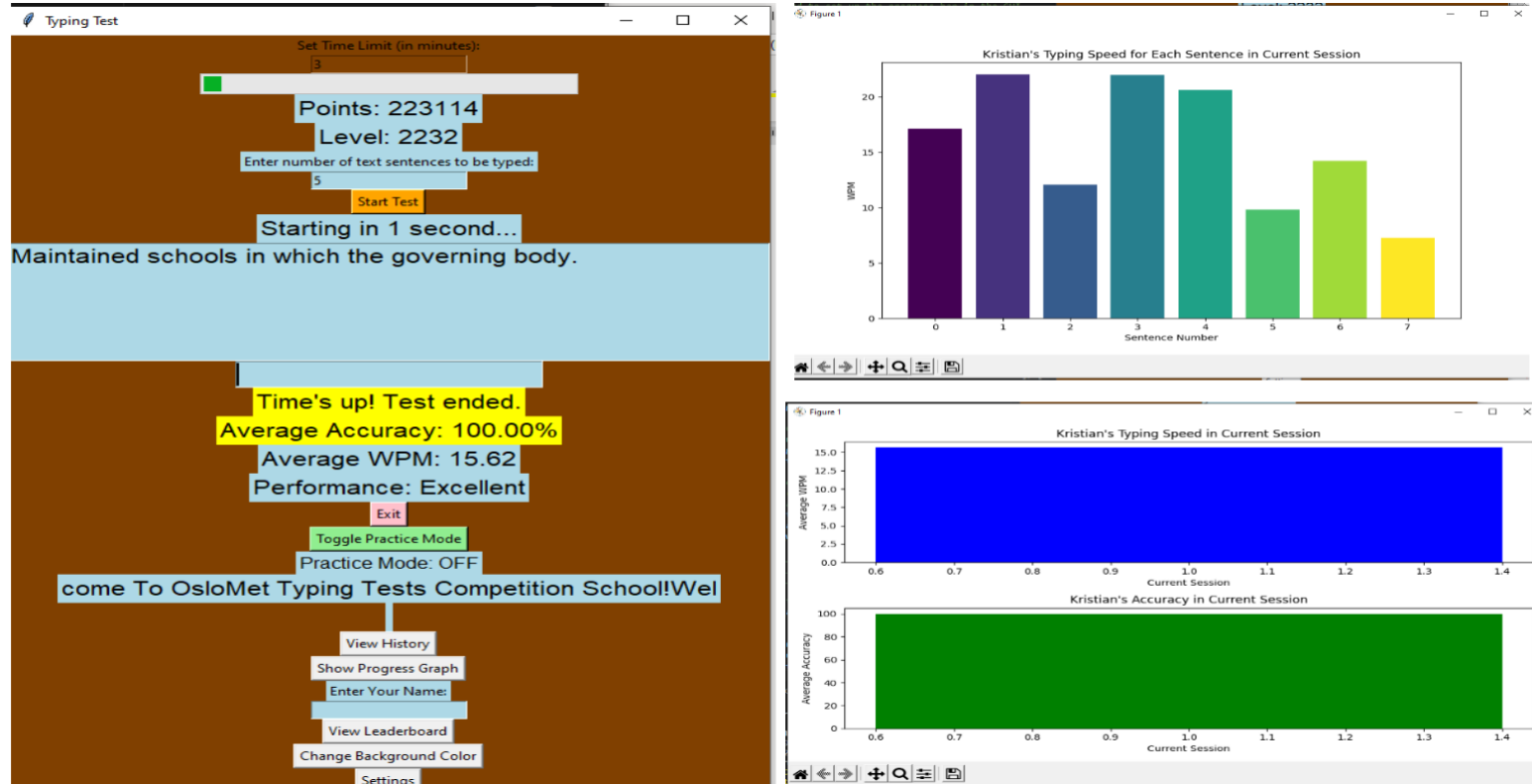


Figure 19: Example 2c: Average Skill Enhancement Session Three

4.3.3 **Example 3: Professional Skill Enhancement:** A professional looking to improve their typing speed for work-related tasks used the application. The improvement in their typing speed and the application's role in this enhancement were documented. Figure

20

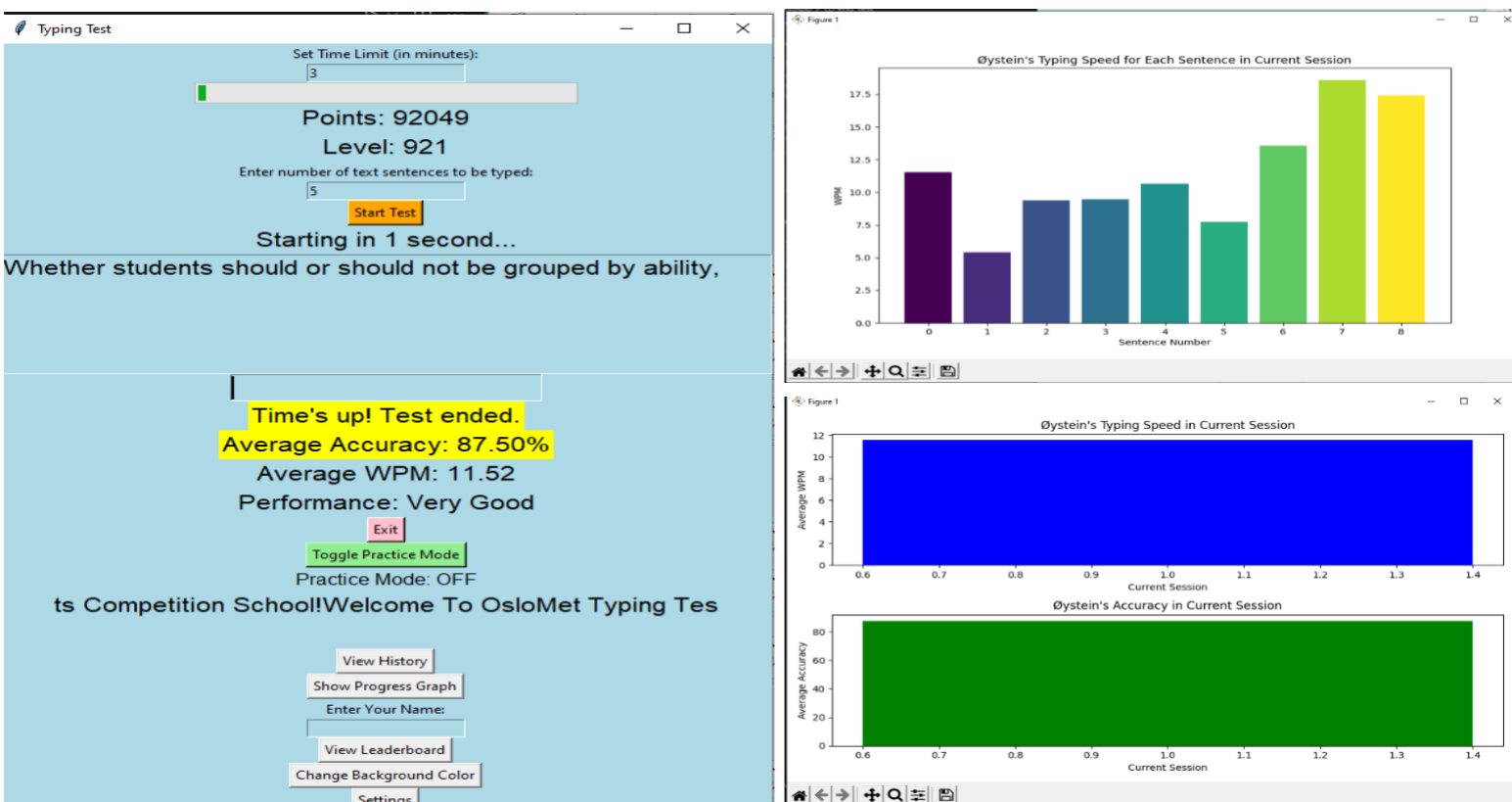


Figure 20: Example 3a: Professional Skill Enhancement Session One

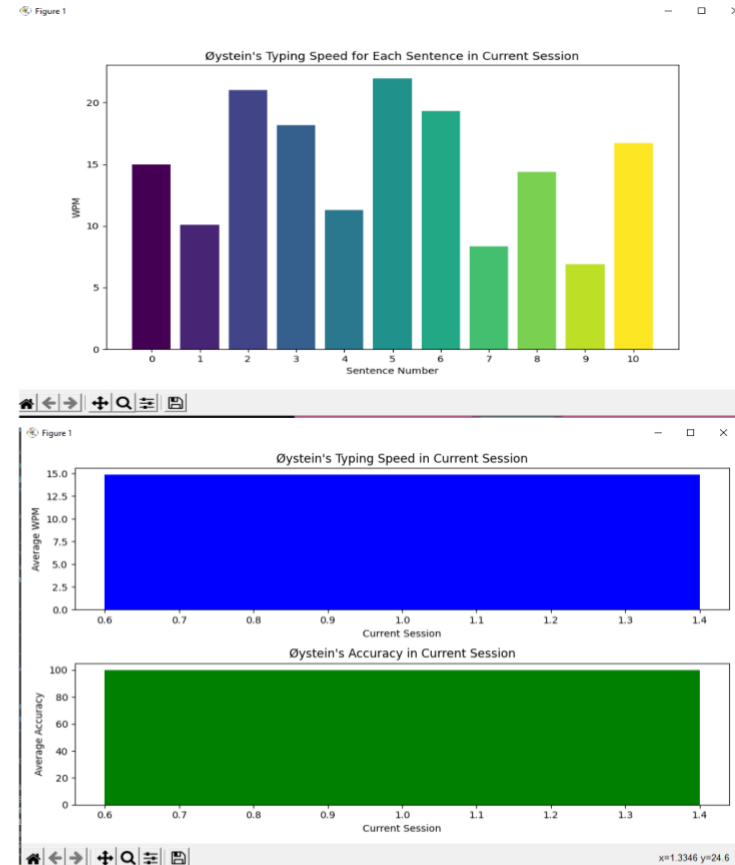
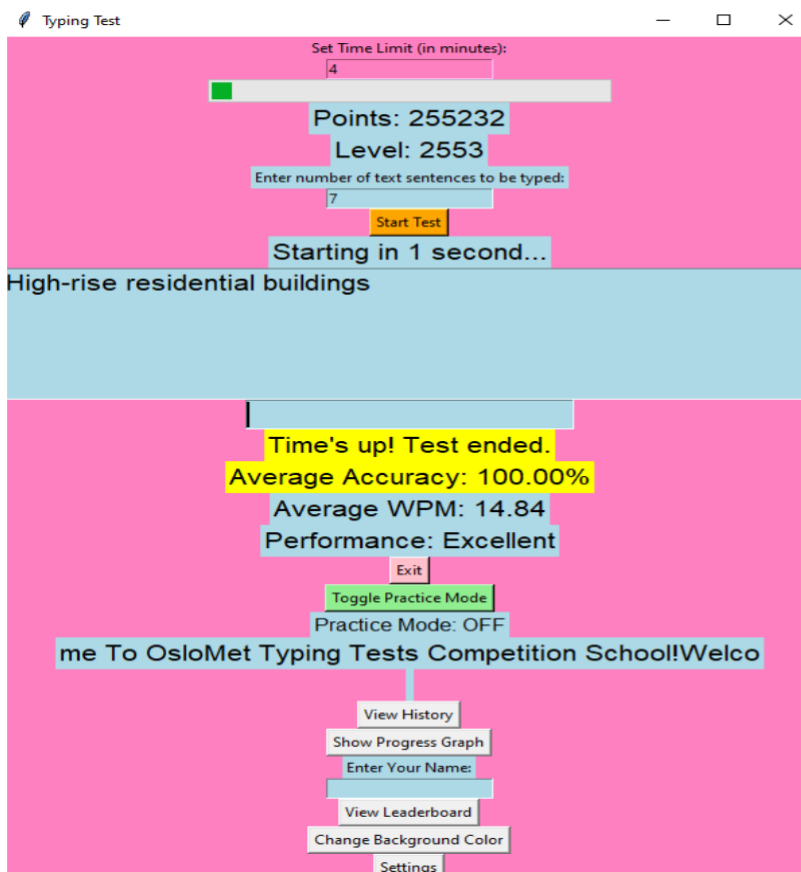


Figure 21: Example 3b: Professional Skill Enhancement Session Two

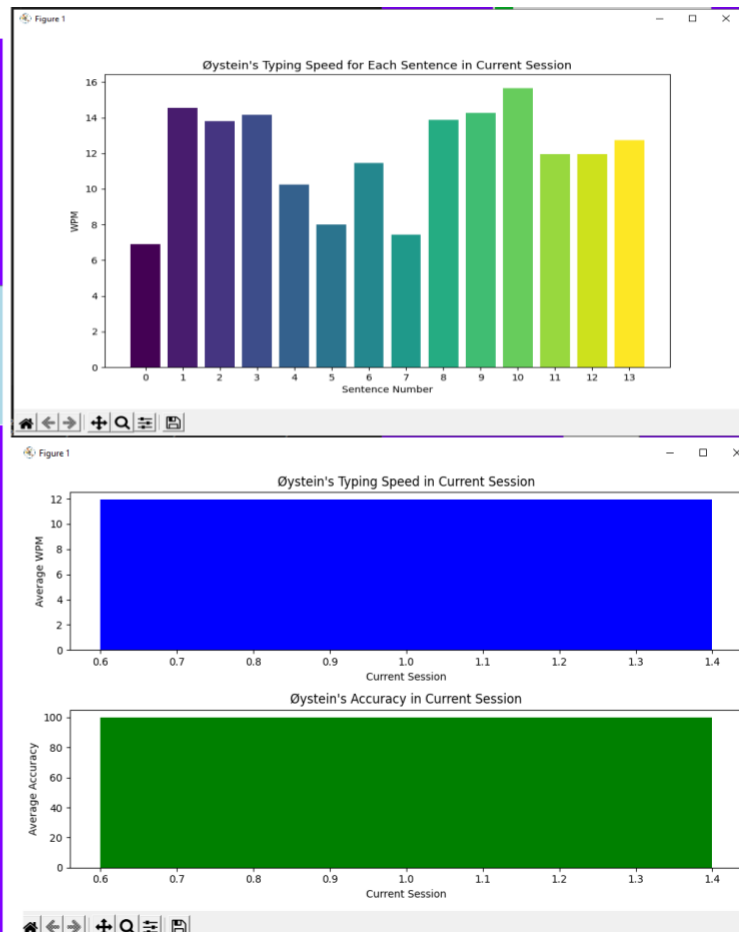
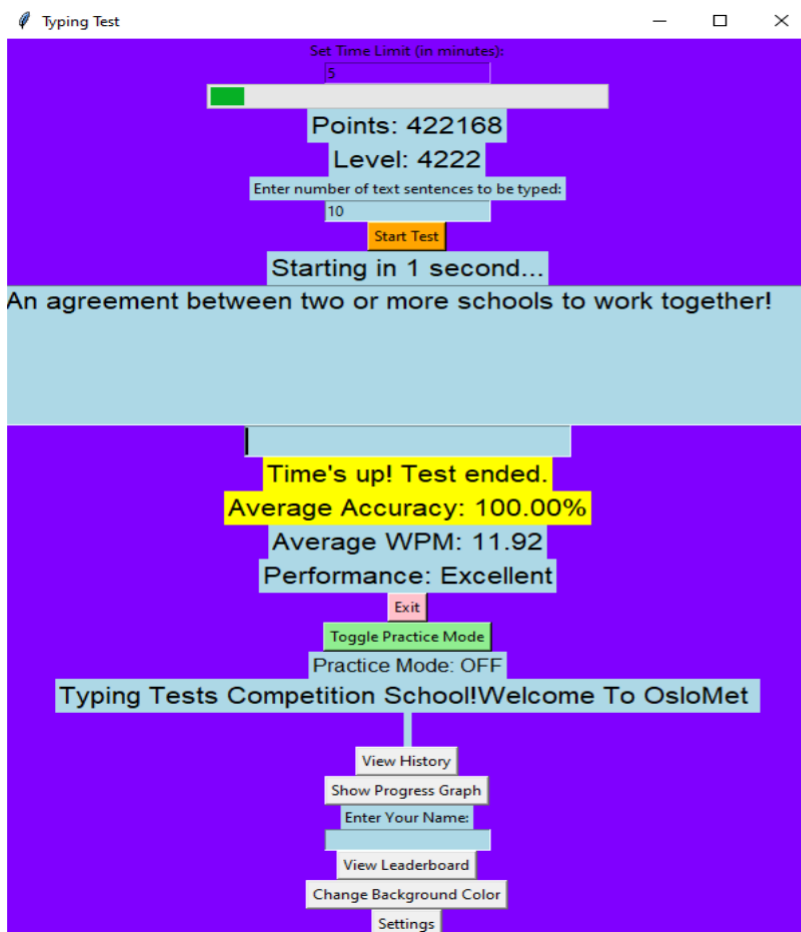


Figure 22: Example 3c: Professional Skill Enhancement Session Three

4.4 Challenges and Resolutions

During the development and testing phases, several challenges were encountered:

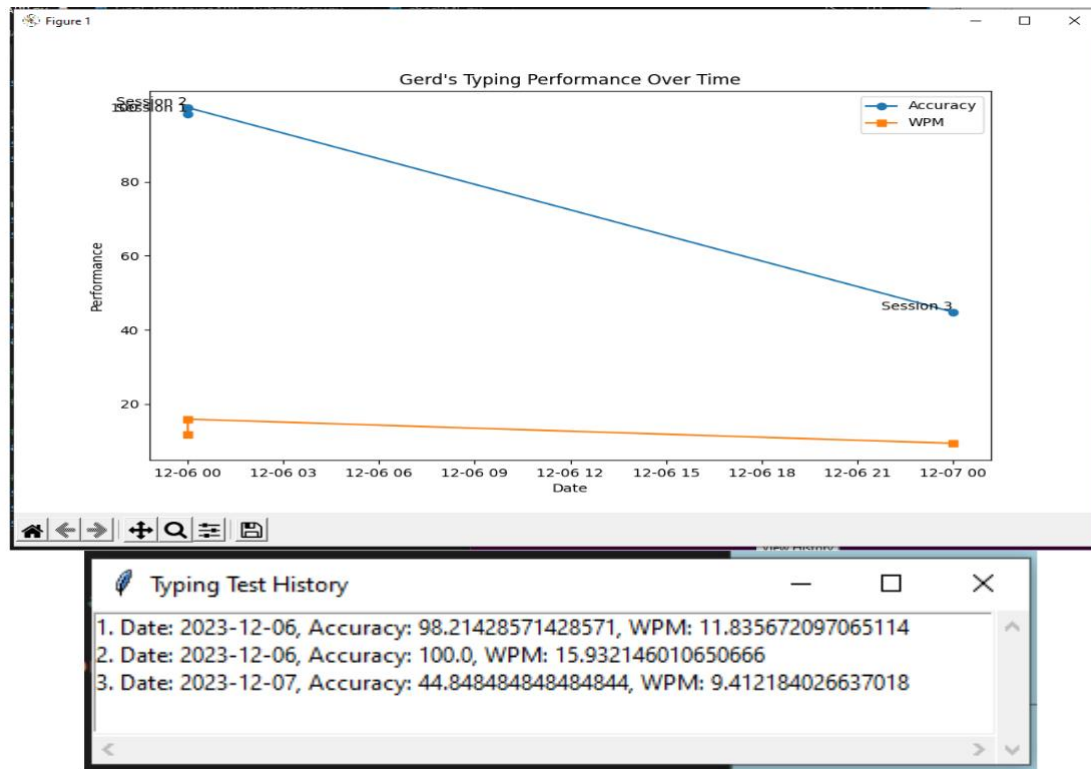
- 4.4.1 **Performance Optimization for Voice Feedback:** Initially, the voice feedback feature caused a slowdown in the application's performance, particularly when pronouncing letters. This issue has been addressed, and the application now efficiently handles voice feedback without a significant impact on typing responsiveness. This improvement ensures that users can enjoy the auditory feedback feature without experiencing lag or delays in the application's response.
- 4.4.2 **Elimination of Redundant Voice Feedback:** The application initially had an issue where it would occasionally repeat the pronunciation of a letter. This redundancy has been resolved, ensuring that each letter is pronounced only once as it is typed. This fix enhances the clarity and usefulness of the voice feedback feature, making the typing experience more seamless and less distracting.
- 4.4.3 **GUI Responsiveness:** Initially, the application faced issues with GUI lag during intensive typing sessions. This was resolved by optimizing the event handling mechanism in Tkinter.
- 4.4.4 **Data Integrity:** Ensuring the accuracy and consistency of performance data was a challenge. Implementing robust error handling and data validation mechanisms addressed this issue.
- 4.4.5 **User Experience:** Feedback from user acceptance testing led to several UI/UX improvements, such as enhanced error highlighting and more intuitive navigation.

4.5 Documentation and Code Comments

Comprehensive documentation and in-code comments were maintained throughout the development process, ensuring that the application is maintainable and scalable.

5 Analysis and Results

The Figure 23 below presents a detailed summarized analysis of results obtained from the above Figure 14, Figure 15, and Figure 16



Gerd's Typing Session Data Analysis	
Typing Session One <ul style="list-style-type: none">Date: 2023-12-06User ID and Name: GerdAverage Accuracy: 98.21%Average Words Per Minute (WPM): 11.84Score: 11,624.32Number of Sentences Typed: 7Session Duration: 2 minutesTotal Points: 80,116Level: 802 <p>Analysis: In this session, Gerd demonstrated exceptional accuracy, nearly perfect, which is indicative of careful and precise typing. However, the average WPM is relatively modest, suggesting a focus on accuracy over speed. The high score and level achieved are reflective of the excellent accuracy. The session was relatively short, but the number of sentences typed in this duration indicates efficient typing.</p>	<ul style="list-style-type: none">Average WPM: 9.41Score: 4,221.22Number of Sentences Typed: 11Session Duration: 4 minutesTotal Points: 245,308Level: 2434 <p>Analysis: This session shows a significant drop in both accuracy and speed. The accuracy level is below half, which could be due to several factors such as increased difficulty of the text, less focus, or perhaps fatigue. The decrease in WPM aligns with the lower accuracy, possibly indicating a struggle with the text. Despite this, the total points and level are still high, which could be attributed to the cumulative performance over the sessions.</p>
Typing Session Two <ul style="list-style-type: none">Date: 2023-12-06User ID and Name: GerdAverage Accuracy: 100%Average WPM: 15.93Score: 15,932.15Number of Sentences Typed: 7Session Duration: 3 minutesTotal Points: 191,637Level: 1917 <p>Analysis: This session shows a remarkable improvement in both accuracy and speed compared to the first session. Achieving 100% accuracy indicates flawless typing. The increase in WPM suggests that Gerd was more confident and quicker while maintaining accuracy. The score and level achieved are significantly higher, reflecting the improved performance. The consistency in the number of sentences typed, despite the longer session duration, might indicate more complex or longer sentences being typed in this session.</p>	<p>Overall Interpretation</p> <p>Gerd shows a fluctuating performance across the sessions. The first two sessions indicate a strong focus on accuracy, with a notable improvement in speed in the second session. However, the third session showed a marked decrease in performance, which could be due to various external or internal factors. The overall high points and level achieved suggest that Gerd generally performs well in typing tasks, with the potential for high accuracy and reasonable speed. The data also indicates that Gerd's performance might be inconsistent, which is an area for potential focus and improvement.</p>
Typing Session Three <ul style="list-style-type: none">Date: 2023-12-07User ID and Name: GerdAverage Accuracy: 44.85%	

Figure 23: Example 1: Learning Typing Skills Results Analysis

Th Figure 24 below presents a detailed summarized analysis of results obtained from the above Figure 17, Figure 18, Figure 19.

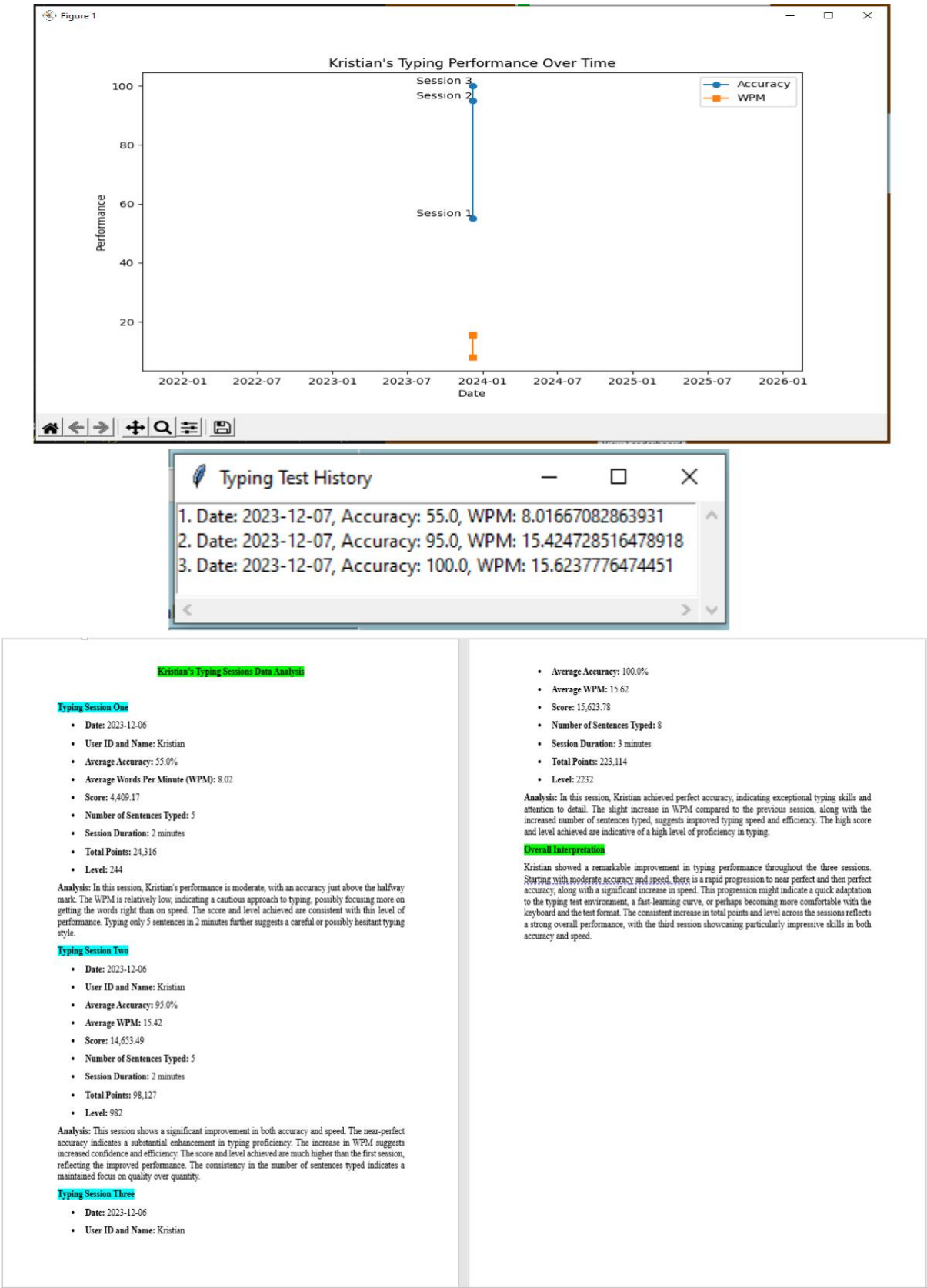


Figure 24: Example 2: Average Skill Enhancement Results Analysis

Th Figure 24 below presents a detailed summarized analysis of results obtained from the above Figure 20, Figure 21, and Figure 22.

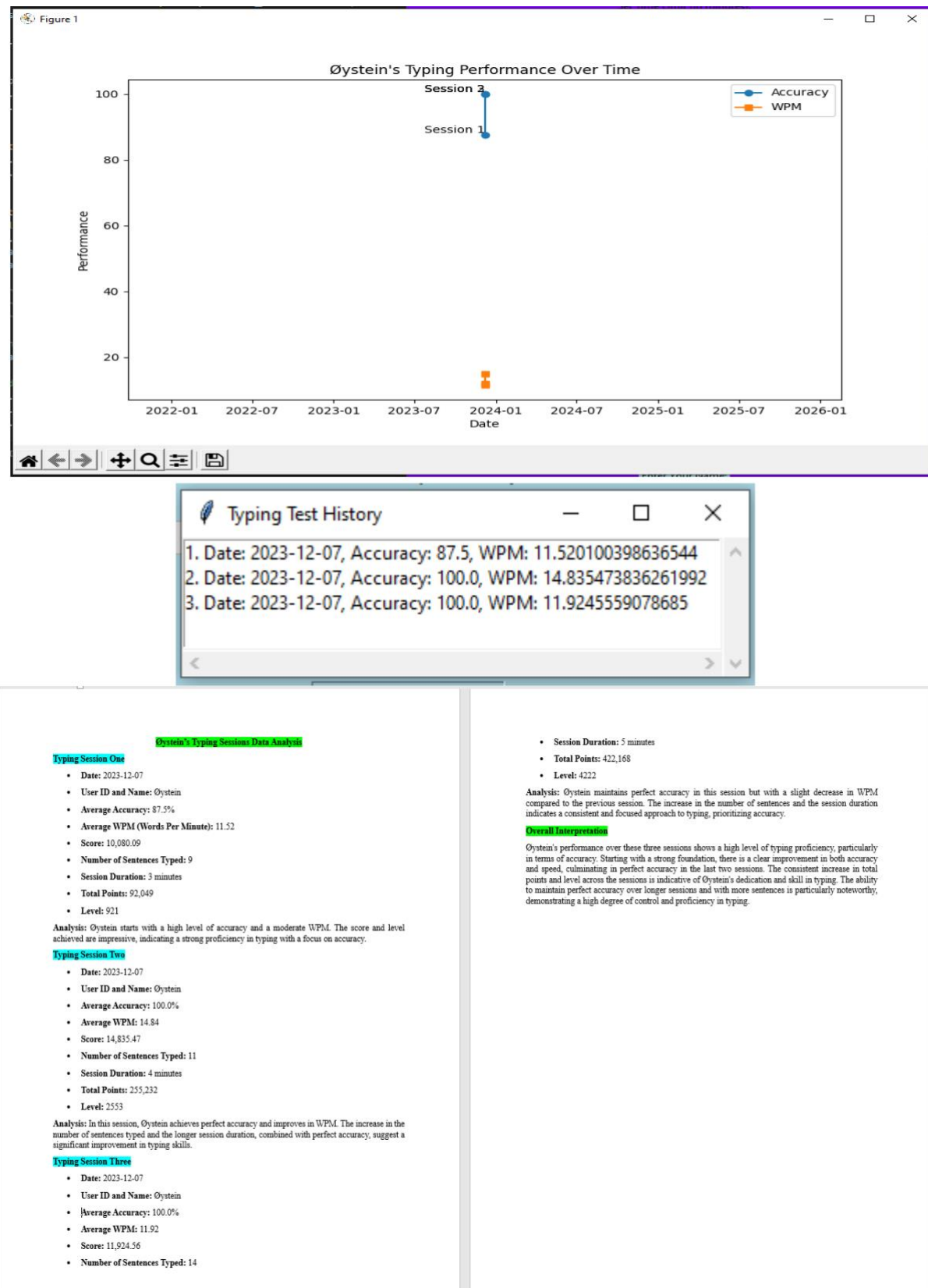


Figure 25: Example 3: Professional Skill Enhancement Results Analysis

5.1 Application Performance

This application's robust performance was evaluated under various user levels as illustrated by the figures above, including different hardware setups and operating systems. It consistently maintained high responsiveness and accuracy in performance metrics. Stress tests showed that the application could handle extended usage without any significant decrease in performance. The real-time feedback mechanisms, such as error highlighting and speed tracking, functioned effectively.

5.2 User Feedback and Improvements

Feedback from users was overwhelmingly positive. They were systematically collected and analyzed. While most users found the application helpful and user-friendly, some suggested enhancements like more personalized feedback mechanisms and the inclusion of different language options for a wider user base.

5.3 Educational Impact

The application's impact was particularly notable in educational settings. Teachers reported its usefulness in typing classes, noting improvements in students' typing skills and their increased engagement with the learning process. The application's adaptability to different skill levels made it a valuable tool in diverse learning environments.

5.4 Future Enhancements

Considering the analysis and user feedback, several enhancements are planned for future versions. These include the development of an AI-driven personal tutor feature, edit button, real-time leaderboard data visualization which would provide more personalized guidance and feedback to users based on their performance history. Printing of a current user's name and selection list options of the set time limit on the GUI. Additionally, more exciting introduction of gamification elements is being considered to make the learning process more engaging and fun.

6 Discussion and Recommendations

The findings show that the Typing Tester Application is successful in helping people type better. Users like the app. It takes the usual parts of a typing test and adds new things like moving text, earning points, going up levels, speaking of words, timing, and keeping track of progress. All these features make learning more fun and effective. The feedback I've gotten and the data I've collected show that the app has a good effect on how well and how eagerly users type. But there's still a chance to add more kinds of content and features to meet the needs of even more users.

The app's design, which is easy to understand and use, makes it a good choice for both beginners and those who already know how to type but want to get better. The real-time feedback on accuracy and speed is especially helpful because it lets users see right away how they're doing and where they can improve. This immediate response helps users stay motivated and focused on their goals.

The speaking feature, which reads out words or sentences, is another unique aspect that sets this application apart. It's not only helpful for those learning to type but also for visually impaired users. This inclusive design approach broadens the app's appeal and usefulness.

Looking ahead, the application could be enhanced by adding more diverse typing exercises, such as different genres of text, to keep users engaged and challenged. Also, integrating more personalized features, like custom difficulty levels or targeted practice sessions based on user performance, could make the app even more effective.

In conclusion, the Typing Tester Application is a well-rounded tool that successfully combines educational value with engaging features. Its ongoing development and potential for new features make it a promising resource for anyone looking to improve their typing skills in a fun and interactive way.

7 Conclusion

The Typing Tester Application is a fantastic tool for improving typing skills. It's designed to be easy and fun to use, with features like earning points, moving up levels, and getting instant feedback on how fast and accurately you type. These features have made it very popular. The app is simple to navigate, and it has special features like moving text, dictating, speaking of words, and detailed charts that show your progress. These make learning to type more fun and keep users interested and happy.

Feedback from users and how well they do on the application show that it helps them get better at typing over time. Compared to other typing programs, this Typing Tester stands out, especially for those who want a fun and interactive way to improve their typing skills.

7.1 Final Thoughts

The Typing Tester Application is a great mix of traditional learning and modern technology. It's good at drawing in users and helping them get better at typing. This shows how important it is to focus on what users need and to keep improving. There is a plan to make this computer application even better in the future, to make sure it stays useful and fun in a world that's always changing and becoming more digital. This application isn't just for now; It will be improved all the time to meet the needs of the future.

8 References

- [1] M. Herold, E. Alant, and J. Bornman, "Typing speed, spelling accuracy, and the use of word-prediction," *South African J. Educ.*, vol. 28, no. 1, pp. 117–134, 2008.
- [2] A. Ikhsananto and S. Sutirman, "The Effectiveness of Rapid Typing Software and Module for Improving 10-Finger Typing Skills," *Din. Pendidik.*, vol. 13, no. 2, pp. 228–237, 2018, doi: 10.15294/dp.v13i2.15540.
- [3] M. Yamaguchi, M. J. C. Crump, and G. D. Logan, "Speed-accuracy trade-off in skilled typewriting: Decomposing the contributions of hierarchical control loops," *J. Exp. Psychol. Hum. Percept. Perform.*, vol. 39, no. 3, pp. 678–699, 2013, doi: 10.1037/a0030512.
- [4] G. D. Yamaguchi, M., Crump, M. J. C., & Logan, "Speed–accuracy trade-off in skilled typewriting: Decomposing the contributions of hierarchical control loops.," p. 23, 2013.
- [5] "analysis of typewriting errors made by students in a second-year typewriting class at Leon High School, Tallahassee, Florida _ DigiNole.pdf."
- [6] L. History, "Student User Guide," *System*, pp. 1–13.
- [7] Dan Cagliano, "The King James Version of the Holy Bible," p. 742, 2004, [Online]. Available: <http://www.davince.com/bible>
- [8] C. With *et al.*, "We believe in a real God, who really cares and who has great plans for you. We'd love for you to come and visit us at our sister website Questions God. Com (<https://www.questionsgod.com>) and see what we're all about here (<https://www.questionsgod.com>).," pp. 1–9.
- [9] The Institute for Local Government, "Glossary of Land Use and Planning Terms," p. 72, 2010, [Online]. Available: www.ca-ilg.org
- [10] S. R. Frey, "A Glossary of Agriculture , Environment , and Sustainable Development," *B. Gloss.*, vol. 661, p. 26, 1996.
- [11] A. F. H. Act, "Glossary 4350.3 REV-1," pp. 1–37, 2003.

- [12] Maryland Cooperative Extension, “Agriculture Terms & Definitions,” pp. 1–6, 2008.
- [13] C. Guide and I. E. Team, “76 Marketing Terms You Should Know (Plus Definitions),” pp. 1–10, 2022.
- [14] “Digital Marketing Terms ,” pp. 1–10.
- [15] A. Terms, “Glossary of Business Terms,” 1998, doi: 10.1201/9781420050561.axa.
- [16] B. English, “English - Unknown - Glossary of business terms”.
- [17] K. H. Hellton, “Mathematical libraries and more Crunch some numbers,” <https://oslomet.instructure.com/courses/26698/files>, pp. 0–19, 2023.
- [18] K. H. Hellton, “Error Handling and Regular Expressions A bit of everything,” <https://oslomet.instructure.com/courses/26698/files>, 2023.
- [19] T. Theodoridis and J. Kraemer, *No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析 Title.*
- [20] K. H. Hellton, “Collections , Strings , and Files Collating information,” <https://oslomet.instructure.com/courses/26698/files>, pp. 0–20, 2023.
- [21] K. H. Hellton, “Libraries for statistics and visualizations Get some data,” <https://oslomet.instructure.com/courses/26698/files>, 2023.
- [22] K. H. Hellton, “Welcome to ACIT4420 Problem-solving with scripting Who are we ?,” <https://oslomet.instructure.com/courses/26698/files>, pp. 0–22, 2023.
- [23] K. H. Hellton, “Higher-dimensional lists and functions Lists, maps and matrices,” <https://oslomet.instructure.com/courses/26698/files>, pp. 0–18, 2023.
- [24] P. Louridas, “Version control,” *IEEE Softw.*, vol. 23, no. 1, pp. 104–107, 2006, doi: 10.1109/MS.2006.32.
- [25] S. Berg, R. Gommers, and C. H. Et.al, “Numpy,” 2023. <https://numpy.org/>
- [26] matplotlib.org, “Pyplot_Tutorial,” matplotlib.org, 2020.
- [27] Python, “The Python Standard Library,” *Python Softw. Found.*, pp. 3–9, 2021.

- [28] P. M. P. Libraries and P. D. F. Extraction, “POWER Lille Gensen Python Libraries for PDF Extraction,” pp. 1–14.
- [29] R. da Silva Leote, “Viridis,” 2019. doi: 10.1145/3359852.3359976.
- [30] E. Bucher *et al.*, “Annot: A Django-based sample, reagent, and experiment metadata tracking system,” *BMC Bioinformatics*, vol. 20, no. 1, 2019, doi: 10.1186/s12859-019-3147-0.

9 Appendices

9.1 Appendix A

NOTE: *The complete codebase for this standalone application is also delivered along with this report on the university examination portal (Inspira). Everything needed to setup this application including the data source file (sentences.txt), user_performance_data.csv, user.csv (all these came with their backup copies) and is zipped. For easy accessibility, the complete code as a single file has also been placed below under the Appendix B*. To access the credentials of the existing users, open the **user.csv** file inside the zipped folder of containing this report and others.

USAGE:

1. Starting the Application:

- Run the application, and the main window will open with various options like Start Test, View History, and Settings.

2. Registering and Logging In:

- New users must register first. Returning users can log in with their credentials to access their personalized data.

3. Conducting a Typing Test:

- Set the desired number of examples and time limit, then start the test. Type the displayed sentences as accurately and quickly as possible.

4. Viewing Results and History:

- After completing a test, view the results immediately. Access the history to see past performances.

5. Analyzing Progress:

- Use the progress graphs to visually track improvements in typing speed and accuracy over time.

6. Competing on the Leaderboard:

- Check the leaderboard to see where you stand among other typists and aim to improve your ranking.

7. Customizing the Experience:

- Adjust the application's theme, font, and text color to suit your preferences for a more comfortable typing experience.

Conclusion: The Typing Tester Application is an effective tool for anyone looking to enhance their typing skills. Its blend of user-friendly design, comprehensive features, and interactive elements makes it suitable for typists of all levels, from beginners to advanced.

9.2 Appendix B

```
""" =====
THE IMPORTATION OF USED PYTHON LIBRARIES
===== """

import csv
import os
from datetime import datetime, timedelta
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
import time
import random
import matplotlib.pyplot as plt
import threading
import numpy as np
import pytttsx3 # Import the pytttsx3 library
import queue
from queue import Queue
from tkinter import colorchooser # Import the colorchooser module

""" =====
THE IMPLEMENTATION OF GLOBAL FUNCTIONS STARTS HERE
===== """

# Function or Method that loads User Credentials
# Returns a dictionary of username-password pairs
def load_user_credentials():
    users = {}
    try:
        # Open the CSV file and read each line
        with open('users.csv', 'r') as file:
            for line in file:
                # Split each line into username and password
                username, password = line.strip().split(',')
                users[username] = password
    except FileNotFoundError:
        pass # File not found, will be created during registration
    return users

# Function to save performance data including user name
def save_performance_data(user_id, user_name, date, accuracy, wpm):
    try:
        score = calculate_score(accuracy, wpm) # Calculate the score
        # Open the file in append mode
        with open('user_performance.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            # Write the data
```



```

        writer.writerow([user_id, user_name, date, accuracy, wpm, score])
    except IOError:
        print("Error in saving performance data")

# Function to load performance data
def load_performance_data(user_id):
    performance_data = []
    try:
        with open('user_performance.csv', 'r') as file:
            reader = csv.reader(file)
            for row in reader:
                if row[0] == user_id:
                    performance_data.append(row)
    except IOError:
        print("Error in loading performance data")
    return performance_data

# Function to load leaderboard data
def load_leaderboard_data():
    leaderboard_data = []
    try:
        with open('user_performance.csv', 'r') as file:
            reader = csv.reader(file)
            for row in reader:
                if len(row) == 6: # Ensure the row has at least 5 columns
                    leaderboard_data.append(row)
    except IOError as e:
        print(f"Error in loading leaderboard data: {e}")
    return sorted(leaderboard_data, key=lambda x: float(x[4]),
reverse=True)[:65] # Top 10 performances based on WPM

# Function to filter data by date
def filter_data_by_date(data, start_date, end_date):
    filtered_data = []
    for row in data:
        date = datetime.strptime(row[1], '%Y-%m-%d')
        if start_date <= date <= end_date:
            filtered_data.append(row)
    return filtered_data

# Function to load text examples from a
file def load_text_examples(file_path):
    with open(file_path, 'r', encoding='utf-8') as file: sentences
        = [line.strip() for line in file if line.strip()]
    return sentences

# Function to calculate typing accuracy
def calculate_accuracy(original, typed):

```

```

# Split the original text and the typed text into
words original_words = original.split()
typed_words = typed.split()

# Initialize a variable to count the number of correct
words correct_word_count = 0

# Iterate over the pairs of original and typed words
for o, t in zip(original_words, typed_words):
    # Increase the count if the words
    match if o == t:
        correct_word_count += 1

# Calculate accuracy as the ratio of correct words to total words in
the original text
# Check if the original text is not empty to avoid division by zero
accuracy = (correct_word_count / len(original_words)) * 100 if
original_words else 0
return accuracy # This should be a value between 0 and 100

# Function to calculate score based on accuracy and WPM
def calculate_score(accuracy, wpm):
    # Example scoring logic: score is 10 times the product of accuracy and WPM
    return 10 * accuracy * wpm

# Function to determine performance remark based on accuracy
def get_performance_remark(accuracy):
    if accuracy >= 0.95:
        return "Excellent"
    elif accuracy >= 0.80:
        return "Very Good"
    elif accuracy >= 0.65:
        return "Good"
    elif accuracy >= 0.55:
        return "Fair"
    elif accuracy >= 0.45:
        return "Poor"
    else:
        return "Needs Improvement"

""" =====
THE IMPLEMENTATION OF GLOBAL FUNCTIONS STARTS HERE
===== """

""" =====
THE IMPLEMENTATION OF THE MAIN CLASS STARTS HERE
===== """
# Main class for the Typing Test Application

```

```

class TypingTestApp:
    """ ===== THE IMPLEMENTATION OF
    LOCAL FUNCTIONS (POPULARLY KNOWN AS METHODS) STARTS HERE
    ===== """
    def __init__(self, user_id, root, sentences):
        self.root = root self.original_sentences =
        sentences.copy() self.sentences =
        sentences self.current_example = None
        self.start_time = None self.typing_speeds
        = []

        self.total_accuracy = [] # Initialize as a list
        self.num_examples = len(sentences) self.practice_mode
        = False self.session_time_limit = 300 # 5 minutes in
        seconds self.session_end_time = None
        self.test_active = False

        # Attributes for user authentication
        self.is_authenticated = False

        # Create a label for displaying the time limit
        self.time_limit_label = tk.Label(root, text="Set Time Limit (in
minutes):", bg='lightblue')
        self.time_limit_label.pack()

        self.time_limit_entry = tk.Entry(root, bg='lightblue')
        self.time_limit_entry.pack()

        self.user_id = user_id # "user123" Example user ID, can be
dynamically set
        self.points = 0
        self.level = 1

        self.last_speech_time = 0
        self.speech_delay = 1 # seconds

        self.session_data = []
        self.session_end_time = time.time() + self.session_time_limit

        # Setting up the GUI elements like labels, buttons, text entry,
        etc. self.root.title("Typing Test")
        self.root.configure(bg='lightblue')

        # Initialize the progress bar
        self.setup_progress_bar()

```

```

        # Display points and level
        self.points_label = tk.Label(root, text=f"Points: {self.points}",
font=("Helvetica", 16), bg='lightblue')
        self.points_label.pack()

        self.level_label = tk.Label(root, text=f"Level: {self.level}",
font=("Helvetica", 16), bg='lightblue')
        self.level_label.pack()

        # User input for number of examples
        self.num_examples_label = tk.Label(root, text="Enter number of text
sentences to be typed:", bg='lightblue')
        self.num_examples_label.pack()

        self.num_examples_entry = tk.Entry(root, bg='lightblue')
        self.num_examples_entry.pack()

        # Start button
        self.start_button = tk.Button(root, text="Start Test",
command=self.start_countdown, bg='orange')
        self.start_button.pack()

        # Countdown label
        self.countdown_label = tk.Label(root, text="", font=("Helvetica", 16),
bg='lightblue')
        self.countdown_label.pack()

        # Text display and entry
        self.text_to_type = tk.Text(root, height=5, width=50,
font=("Helvetica", 16), bg='lightblue')
        self.text_to_type.pack()
        self.text_to_type.tag_configure("error", background="white",
foreground="red") # Error highlighting style
        self.entry = tk.Entry(root, font=("Helvetica", 16), bg='lightblue')
        self.entry.pack()
        self.entry.bind("<KeyRelease>", self.on_key_release) # Bind key
release event
        self.text_to_type.bind("<KeyRelease>", self.on_key_release)
        self.entry.bind("<Return>", self.submit_text)

        # Results display
        self.results_label = tk.Label(root, text="", font=("Helvetica", 16),
bg='yellow')
        self.results_label.pack()

        self.average_accuracy_label = tk.Label(root, text="",
font=("Helvetica", 16), bg='yellow')

```

```

        self.average_accuracy_label.pack()

        self.average_wpm_label = tk.Label(root, text="", font=("Helvetica",
16), bg='lightblue')
        self.average_wpm_label.pack()

        self.performance_remark_label = tk.Label(root, text="",
font=("Helvetica", 16), bg='lightblue')
        self.performance_remark_label.pack()

        # Exit button
        self.exit_button = tk.Button(root, text="Exit", command=root.destroy,
bg='pink')
        self.exit_button.pack()

        # Practice Mode Toggle
        self.practice_mode_button = tk.Button(root, text="Toggle Practice
Mode", command=self.toggle_practice_mode, bg='lightgreen')
        self.practice_mode_button.pack()
        self.practice_mode_label = tk.Label(root, text="Practice Mode: OFF",
font=("Helvetica", 12), bg='lightblue')
        self.practice_mode_label.pack()

        # Animated Text Label
        self.animated_text_label = tk.Label(root, text="Welcome To OsloMet
Typing Tests Competition School!", font=("Helvetica", 16), bg='lightblue')
        self.animated_text_label.pack()
        self.animate_text()

        # Initialize the text-to-speech engine
        self.speech_queue = queue.Queue()
        self.speech_engine = pyttsx3.init()
        self.speech_engine.setProperty('rate', 150) # Set speech rate
        self.currently_speaking = None
        # Start the speech processing thread
        threading.Thread(target=self.process_speech_queue,
daemon=True).start()

        # Add a label for the countdown
        self.session_time_label = tk.Label(root, text="", font=("Helvetica",
16), bg='lightblue')
        self.session_time_label.pack()

        # Add buttons for viewing history and progress graph
        self.view_history_button = tk.Button(self.root, text="View History",
command=self.show_history)
        self.view_history_button.pack()

```

```

        self.show_graph_button = tk.Button(self.root, text="Show Progress
Graph", command=self.show_progress_graph)
        self.show_graph_button.pack()

        # Create a label and entry for the user name
        self.user_name_label = tk.Label(self.root, text="Enter Your Name:",
bg='lightblue')
        self.user_name_label.pack()

        self.user_name_entry = tk.Entry(self.root, bg='lightblue')
        self.user_name_entry.pack()

        # Add a button to view the leaderboard
        self.leaderboard_button = tk.Button(self.root, text="View
Leaderboard", command=self.show_leaderboard)
        self.leaderboard_button.pack()

        # Add a button to change the background color
        self.color_button = tk.Button(root, text="Change Background Color",
command=self.choose_background_color)
        self.color_button.pack()

        # Add a button to open the settings window
        self.settings_button = tk.Button(root, text="Settings",
command=self.create_settings_window)
        self.settings_button.pack()

        # Prompt for user login
        self.login_window()

    def get_current_word_or_sentence(self):
        # Implement the logic to extract the current word or sentence
        # For now, let's just return a placeholder string
        return "current word or sentence"

    # Method for User's Registration
    Form def on_register(self):
        entered_username = self.username_entry.get().strip()
        entered_password = self.password_entry.get().strip()
        # Verify credentials (username and password)
        if entered_username and entered_password:
            # Load existing users
            users = load_user_credentials()

            if entered_username in users:
                tk.messagebox.showerror("Registration Failed", "Username
already exists.")

```

```

        return

        # Append new user to the file
        with open('users.csv', 'a') as file:
            file.write(f"{entered_username},{entered_password}\n")

        tk.messagebox.showinfo("Registration", "Registration
successful. You can now log in.")
    else:
        # Show error message for invalid credentials
        tk.messagebox.showerror("Registration Failed", "Username
and password cannot be empty.")

# Method to collect both username and userID
def login_window(self):
    self.login_win = tk.Toplevel(self.root)
    self.login_win.title("Login")
    self.login_win.geometry("300x200")

    tk.Label(self.login_win, text="Welcome to our typing tester practicing
system").pack()
    tk.Label(self.login_win, text="Username:").pack()
    self.username_entry = tk.Entry(self.login_win)
    self.username_entry.pack()

    tk.Label(self.login_win, text="Password:").pack() # Changed from User
ID to Password
    self.password_entry = tk.Entry(self.login_win, show="*")
    self.password_entry.pack()

    tk.Button(self.login_win, text="Login", command=self.on_login).pack()
    tk.Button(self.login_win, text="Register",
command=self.on_register).pack() # Registration button

# This method is called when the user clicks the login button. It
stores the username and user ID and then closes the login window.
def on_login(self):
    entered_username = self.username_entry.get().strip()
    entered_password = self.password_entry.get().strip()

    # Load existing users
    users = load_user_credentials()
    # Verify credentials (You need to implement this method)
    if entered_username in users and users[entered_username] ==
entered_password:
        self.user_name = entered_username
        self.user_id = entered_username # Assuming username as user ID
for simplicity

```

```

        self.is_authenticated = True
        # Close the login window and display a welcome message
        self.login_win.destroy() tk.messagebox.showinfo("Welcome",
        f"Welcome to the system,
{self.user_name}!")
    else:
        # Show error message for invalid credentials
        tk.messagebox.showerror("Login Failed", "Invalid username or
password.")

def verify_credentials(self, username, user_id):
    # Implement your credential verification logic here
    # For now, it just returns True for demonstration
    purposes return True

# Method to speak the given text
def speak_text(self, text):
    # Check if the text is already in the queue or being spoken
    if not self.is_text_in_queue(text) and self.currently_speaking !=
text:
        # Add text to the queue if it's not too full
        if self.speech_queue.qsize() < 5: # Adjust the threshold as
needed
            self.speech_queue.put(text)

def is_text_in_queue(self, text):
    # Check if the text is already in the
queue with self.speech_queue.mutex:
    return text in self.speech_queue.queue

# Method to handle functionality for speaking our the
text def _speak(self, text):
    while True:
        text = self.speech_queue.get()
        self.speech_engine.say(text)
        self.speech_engine.runAndWait()
        self.speech_queue.task_done()

# This method Process the speech queue continuously.
def process_speech_queue(self):
    while True:
        text = self.speech_queue.get()
        self.currently_speaking = text
        self.speech_engine.say(text)
        self.speech_engine.runAndWait()
        self.currently_speaking = None
        # Clear the queue to prioritize latest
input with self.speech_queue.mutex:

```



```

        self.speech_queue.queue.clear()
        self.speech_queue.task_done()

# Method to animate text in the
GUI def animate_text(self):
    def rotate_text(text, i): display_text
        = text[i:] + text[:i]
        self.animated_text_label.config(text=display_text)
        self.root.after(300, lambda: rotate_text(text, (i + 1) %
len(text)))
        rotate_text("Welcome To OsloMet Typing Tests Competition School!", 0)

# Method to start the countdown before the test
begins def start_countdown(self):
    self.countdown_label.config(text="Starting in 3 seconds...")
    self.root.after(1000, lambda:
self.countdown_label.config(text="Starting in 2 seconds..."))
    self.root.after(2000, lambda:
self.countdown_label.config(text="Starting in 1 second..."))
    self.root.after(3000, self.prepare_test)

# Method called when a key is released in the text entry
# It checks for typing errors and updates the display
accordingly def on_key_release(self, event):
    typed_text = self.entry.get()
    self.highlight_errors(typed_text)

# Speak the last character typed, if
any if typed_text:
    last_char = typed_text[-1]
    self.speak_text(last_char)

# Speak the last character typed, if
any current_time = time.time()
if current_time - self.last_speech_time > self.speech_delay:
    self.speak_text(event.char)
    self.last_speech_time = current_time

# Example: Speak only the current word or sentence
current_text = self.get_current_word_or_sentence()
if current_text:
    self.speak_text(current_text)

# Method to highlight errors in the typed
text def highlight_errors(self, typed_text):
    self.text_to_type.tag_remove("error", "1.0", tk.END)
    min_length = min(len(self.current_example), len(typed_text))
    for i in range(min_length):

```

```

        if self.current_example[i] != typed_text[i]:
            self.text_to_type.tag_add("error", f"1.{i}", f"1.{i+1}")

        self.text_to_type.tag_remove("error", "1.0", tk.END)
        for i, (original_char, typed_char) in
enumerate(zip(self.current_example, typed_text)):
            if original_char != typed_char:
                self.text_to_type.tag_add("error", f"1.{i}", f"1.{i+1}")
                self.text_to_type.tag_config("error", background="yellow",
foreground="red")

# Method to prepare the test by setting the number of
examples def prepare_test(self):
    try:
        new_num_examples = int(self.num_examples_entry.get())
        new_time_limit = int(self.time_limit_entry.get()) * 60 # Convert
minutes to seconds
        if new_num_examples <= 0 or new_time_limit <= 0:
            raise ValueError
    except ValueError:
        self.results_label.config(text="Please enter a valid number of
examples and time limit.")
        return

    if not self.test_active:
        self.session_time_limit = new_time_limit
        self.session_end_time = time.time() + self.session_time_limit
        self.test_active = True
        self.start_test()
    else:
        self.num_examples += new_num_examples
        self.next_example()

# Method to start the typing
test def start_test(self):
    # Reset session data and other variables at the start of each
test self.typing_speeds = []
    self.total_accuracy = [] # Initialize as an empty list
    self.session_data = []
    self.test_active = True
    self.session_end_time = time.time() +
self.session_time_limit self.next_example()
    self.check_session_time()

# Method to check if the session time limit has been reached
def check_session_time(self):
    if not self.practice_mode: # Only check time if not in practice mode
        remaining_time = max(int(self.session_end_time - time.time()), 0)

```

```

        self.session_time_label.config(text=f"Time remaining:
{remaining_time} seconds")
        if remaining_time <= 0:
            if not self.practice_mode:
                if self.sentences:
                    self.record_session_data()
                    self.end_test(time_up=True)
            else:
                self.root.after(1000, self.check_session_time)

# Method record_session_data to handle the session data
recording def record_session_data(self):
    average_wpm = sum(self.typing_speeds) / len(self.typing_speeds) if
self.typing_speeds else 0
    average_accuracy = sum(self.total_accuracy) / len(self.total_accuracy)
if self.total_accuracy else 0
    self.session_data.append({
        'session_number': len(self.session_data) + 1,
        'average_wpm': average_wpm,
        'average_accuracy': average_accuracy
    })

# Method to display the next example for
typing def next_example(self):
    if not self.sentences or time.time() >= self.session_end_time:
        self.end_test()
        return

    self.current_example = self.sentences.pop()
    self.text_to_type.delete("1.0", tk.END)
    self.text_to_type.insert("1.0", self.current_example)
    self.entry.focus_set()
    self.start_time = time.time()

    self.current_example = self.sentences.pop()
    self.text_to_type.delete("1.0", tk.END)
    self.text_to_type.insert("1.0", self.current_example)
    self.entry.focus_set()
    self.start_time = time.time()

# Speak the text
self.speak_text(self.current_example)

completed_examples = self.num_examples - len(self.sentences)
self.update_progress_bar(completed_examples, self.num_examples)

# Method to submit the typed text and calculate
performance def submit_text(self, event=None):

```

```

        if not self.test_active:
            return

        end_time = time.time()
        time_taken = end_time - self.start_time
        typed_text = self.entry.get()
        accuracy = calculate_accuracy(self.current_example, typed_text) #
Calculate accuracy for the current sentence
        wpm = len(typed_text.split()) / (time_taken / 60) # Calculate WPM for
the current sentence
        self.typing_speeds.append(wpm)
        self.total_accuracy.append(accuracy) # Append accuracy for each
sentence

        # wpm = len(self.current_example.split()) / (time_taken / 60)

        # Calculate points (example calculation, you can modify
it) points_earned = int(wpm * accuracy * 10) # Example
formula self.points += points_earned
        self.points_label.config(text=f"Points: {self.points}")
        # Update level based on points
        self.level = self.points // 100 + 1 # Example: level up every 100
points
        self.level_label.config(text=f"Level: {self.level}")

        if self.practice_mode:
            self.results_label.config(text=f"Instant Feedback - Accuracy:
{accuracy:.2%}, WPM: {wpm:.2f}")

        #if not self.practice_mode:

        self.entry.delete(0, tk.END)

        if not self.sentences or (self.practice_mode and len(self.sentences)
== 1):
            self.sentences = self.original_sentences.copy() # Reset sentences
for continuous practice

        self.next_example()

        # Method to set up the progress bar in the
GUI def setup_progress_bar(self):
            self.progress_bar = ttk.Progressbar(self.root, orient="horizontal",
length=300, mode="determinate")
            self.progress_bar.pack()

```

```

# Method to update the progress bar based on current progress
def update_progress_bar(self, current_length, total_length):
    self.progress_bar['value'] = (current_length / total_length) * 100
    self.root.update_idletasks()

# Method to end the test and display results
def end_test(self, time_up=False):
    # Calculate and display average accuracy and WPM
    self.test_active = False
    average_accuracy = sum(self.total_accuracy) / len(self.total_accuracy)
    if self.total_accuracy else 0
    average_wpm = sum(self.typing_speeds) / len(self.typing_speeds) if
self.typing_speeds else 0

    #average_accuracy = self.total_accuracy / len(self.typing_speeds) if
self.typing_speeds else 0
    #average_accuracy = np.mean(self.total_accuracy) if
self.total_accuracy else 0 # Calculate average accuracy
    #average_wpm = np.mean(self.typing_speeds) if self.typing_speeds else
0

    # Convert average accuracy to a decimal
    average_accuracy_decimal = average_accuracy / 100

    # Update the labels for accuracy, WPM, and performance remark
    self.average_accuracy_label.config(text=f"Average Accuracy:
{average_accuracy:.2f}%")
    self.average_wpm_label.config(text=f"Average WPM: {average_wpm:.2f}")
    self.performance_remark_label.config(text=f"Performance:
{get_performance_remark(average_accuracy_decimal)}")

    self.session_time_label.config(text="")
    self.display_bar_chart()

    # Save performance data
    date = datetime.now().strftime('%Y-%m-%d')
    user_name = self.user_name_entry.get() # Get the name entered by the
user
    if not user_name: # Check if the user name is not empty
        # Use the username stored in self.user_name
        user_name = self.user_name if self.user_name else "Anonymous" #
Default name if no name is entered
    print(f"Debug: user_id={self.user_id}, user_name={user_name},
date={date}, average_accuracy={average_accuracy}, average_wpm={average_wpm}")
    save_performance_data(self.user_id, user_name, date, average_accuracy,
average_wpm)

    # Record session data for the current session

```

```

self.session_data.append({
    'session_number': len(self.session_data) + 1,
    'average_wpm': average_wpm,
    'average_accuracy': average_accuracy
})

# Record session data
if not self.practice_mode:
    self.record_session_data()

if time_up:
    self.results_label.config(text="Time's up! Test ended.")
    self.display_session_graphs()

# Check if typing time duration and display graphs upon completion of
the total time.
def check_time_and_display_graphs(self):
    if time.time() >= self.session_end_time:
        self.display_session_graphs()

# Display separate bar graphs for each session
def display_session_graphs(self):
    # Display graphs only for the current
    session if not self.session_data:
        print("No session data to
        display") return

    # Get data for the current session
    current_session = self.session_data[-1]
    plt.figure(figsize=(10, 6))

    # Typing Speed for the Current
    Session plt.subplot(2, 1, 1)
    plt.bar(1, current_session['average_wpm'], color='blue')
    plt.xlabel('Current Session')
    plt.ylabel('Average WPM')
    plt.title(f"{self.user_name}'s Typing Speed in Current Session")

    # Accuracy for the Current Session
    plt.subplot(2, 1, 2)
    plt.bar(1, current_session['average_accuracy'], color='green')
    plt.xlabel('Current Session')
    plt.ylabel('Average Accuracy')
    plt.title(f"{self.user_name}'s Accuracy in Current Session")

    plt.tight_layout()
    plt.show()

```

```

# Method to toggle practice mode
def toggle_practice_mode(self):
    self.practice_mode = not self.practice_mode
    mode_text = "ON" if self.practice_mode else "OFF"
    self.practice_mode_label.config(text=f"Practice Mode: {mode_text}")

# Method to show user's typing history with improved scrollbar and listbox
def show_history(self):
    if not self.is_authenticated:
        tk.messagebox.showerror("Access Denied", "Please log in to view history.")
        return
    performance_data = load_performance_data(self.user_id)
    print(performance_data) # Add this line to inspect the performance data

    history_window = tk.Toplevel(self.root)
    history_window.title("Typing Test History")

    # Create a frame to hold the listbox and scrollbar
    frame = tk.Frame(history_window)
    frame.pack(fill=tk.BOTH, expand=True)

    # Create a scrollbar
    scrollbar_vertical = tk.Scrollbar(frame, orient="vertical")
    scrollbar_horizontal = tk.Scrollbar(frame, orient="horizontal")

    # Create a Listbox to display the history
    history_listbox = tk.Listbox(frame,
yscrollcommand=scrollbar_vertical.set,
xscrollcommand=scrollbar_horizontal.set, width=50)

    for i, (user_id, user_name, date, accuracy, wpm, score) in
enumerate(performance_data):
        history_listbox.insert(tk.END, f"{i+1}. Date: {date}, Accuracy:
{accuracy}, WPM: {wpm}")

    # Pack the scrollbar and listbox
    scrollbar_vertical.pack(side=tk.RIGHT, fill=tk.Y)
    scrollbar_horizontal.pack(side=tk.BOTTOM, fill=tk.X)
    history_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

    # Configure the scrollbars to control the listbox
    scrollbar_vertical.config(command=history_listbox.yview)
    scrollbar_horizontal.config(command=history_listbox.xview)

# Method to show progress graph

```

```

def show_progress_graph(self):
    if not self.is_authenticated:
        tk.messagebox.showerror("Access Denied", "Please log in to view
progress graphs.")
        return
    performance_data = load_performance_data(self.user_id)
    print(performance_data) # Add this line to inspect the performance
data
    dates = []
    accuracies = []
    wpms = []

    for _, _, date, accuracy, wpm, _ in performance_data:
        dates.append(datetime.strptime(date, '%Y-%m-%d'))
        accuracies.append(float(accuracy))
        wpms.append(float(wpm))

    plt.figure(figsize=(10, 6))
    plt.plot(dates, accuracies, marker='o', label='Accuracy')
    plt.plot(dates, wpms, marker='s', label='WPM')
    plt.xlabel('Date')
    plt.ylabel('Performance')
    plt.title(f"{self.user_name}'s Typing Performance Over Time")
    plt.legend()

    # Adding session labels
    for i, date in enumerate(dates):
        plt.text(date, max(accuracies[i], wpms[i]), f"Session {i+1}",
ha='right', va='bottom')

    plt.show()

# Method to display bar chart for current
session def display_bar_chart(self):
    if not self.typing_speeds:
        return

    plt.figure(figsize=(10, 6))
    # Create an array for the x-axis positions
    x_positions = np.arange(len(self.typing_speeds))
    # Plot a bar for each sentence's typing speed
    plt.bar(x_positions, self.typing_speeds,
color=plt.cm.viridis(np.linspace(0, 1, len(self.typing_speeds))))

    plt.xlabel('Sentence Number')
    plt.ylabel('WPM')
    plt.title(f"{self.user_name}'s Typing Speed for Each Sentence
in Current Session")

```



```

plt.xticks(x_positions) # Set the x-ticks to correspond to sentence
numbers
plt.show()

# Function to display the leaderboard
def show_leaderboard(self):
    if not self.is_authenticated:
        tk.messagebox.showerror("Access Denied", "Please log in to view
the leaderboard.")
        return
    try:
        leaderboard_window = tk.Toplevel(self.root)
        leaderboard_window.title("Leaderboard")

        columns = ('User ID', 'User Name', 'Date', 'Accuracy', 'WPM',
'Score')

        leaderboard_tree = ttk.Treeview(leaderboard_window,
columns=columns, show='headings')

        for col in columns:
            leaderboard_tree.heading(col, text=col)

        leaderboard_data = load_leaderboard_data()

        for row in leaderboard_data:
            # Ensure that each column gets the correct data
            leaderboard_tree.insert('', tk.END, values=row) # Insert the
entire row

        leaderboard_tree.pack(expand=True, fill='both')
    except Exception as e:
        tk.messagebox.showerror("Error", f"An error occurred while loading
the leaderboard: {e}")

def choose_background_color(self):
    # Open the color chooser dialog
    color_code = colorchooser.askcolor(title="Choose background color")[1]
    if color_code:
        self.apply_background_color(color_code)

def apply_background_color(self, color):
    # Apply the selected color to the background of the GUI
    elements self.root.configure(bg=color)
    self.time_limit_label.configure(bg=color)
    self.time_limit_entry.configure(bg=color)

def create_settings_window(self):
    settings_window = tk.Toplevel(self.root)

```

```

settings_window.title("Customize Interface")

# Theme Option
tk.Label(settings_window, text="Select Theme:").pack()
self.theme_var = tk.StringVar()
themes = ["Light", "Dark"]
for theme in themes:
    tk.Radiobutton(settings_window, text=theme,
variable=self.theme_var, value=theme, command=self.apply_theme).pack()

# Font Option
tk.Label(settings_window, text="Select Font:").pack()
self.font_var = tk.StringVar()
fonts = ["Helvetica", "Arial", "Times New Roman"]
for font in fonts:
    tk.Radiobutton(settings_window, text=font, variable=self.font_var,
value=font, command=self.apply_font).pack()

# Color Option
tk.Label(settings_window, text="Select Text Color:").pack()
self.color_var = tk.StringVar()
colors = ["Black", "Blue", "Red"]
for color in colors:
    tk.Radiobutton(settings_window, text=color,
variable=self.color_var, value=color, command=self.apply_color).pack()

def apply_theme(self):
    theme = self.theme_var.get()
    if theme == "Dark":
        self.root.configure(bg='gray')
        # Apply dark theme to other widgets...
    else:
        self.root.configure(bg='lightblue')
        # Apply light theme to other widgets...

def apply_font(self):
    font = self.font_var.get()
    # Apply font to widgets...
    self.text_to_type.configure(font=(font, 16))
    self.entry.configure(font=(font, 16))

def apply_color(self):
    color = self.color_var.get()
    # Apply color to widgets...
    self.text_to_type.configure(fg=color)
    self.entry.configure(fg=color)

```

```

""" =====THE IMPLEMENTATION OF
LOCAL FUNCTIONS (POPULARLY KNOWN AS METHODS) STOPS HERE
===== """

```

```

""" =====
THE IMPLEMENTATION OF THE MAIN CLASS STOPS HERE
===== """

```

```

""" =====THE
IMPLEMENTATION EXECUTION OF THIS MAIN CLASS STARTS HERE
===== """

# Main function to run the application
def main():
    root = tk.Tk()
    sentences = load_text_examples("sentences.txt")
    app = TypingTestApp(None, root, sentences) # Initialize with None or some
placeholder
    root.mainloop()

if __name__ == "__main__":
    main()

```

```

""" =====THE
IMPLEMENTATION EXECUTION OF THE MAIN CLASS STOPS HERE
===== """

```