Programming Fundamentals Report

WannaHang @ SIT
(PYTHON GROUP PROJECT)

.

Team Members:

1. Muhd Wafiyuddin
2. <Team Member 2>
3. <Team Member 3>
4. <Team Member 4>
5. <Team Member 5>

I. Abstract

Each year, many freshmen enrolling at SIT are returning to pursue their degrees after a break, such as National Service and gap years. These students often face challenges in adjusting to the sudden overload of information and keeping up with their studies. Our focus is specifically on ICT students, and we recognize that this is a common issue many still encounter.

To address this, we've designed an informal assessment in the form of a game, Hangman, aimed at helping ICT students review and learn content related to the modules they are enrolled in during their first trimester at SIT.

II. Introduction

The "WannaHang @ SIT" initiative is designed to support freshmen students enrolled in the ICT IS/SE programs during their first trimester at SIT. This interactive quiz platform offers a fun and accessible way for students to gauge their readiness for their upcoming final exams. It also provides the chance to win a $20 Grab voucher upon completing a Hangman-style quiz.

The main objective of this project is to help new students familiarize themselves with the level of academic rigour they will encounter during their first exam week. The game features questions covering key topics from the modules students take in their first trimester. By engaging with

these quizzes, students can assess their strengths and identify areas for improvement, ensuring they are better prepared for their exams.

This approach not only helps reduce exam-related stress but also fosters a sense of community and encourages healthy academic competition among SITizens.

Completed Tasks:

- Basic game logic (Login, Input & Output Validation, Completion)
- User Interface (GUI layout including buttons, labels, Tkinter or Pygame)
- Difficulty Level/Timer
- Score tracking
- Music (Main Menu, Easy, Medium, Hard Difficulty, Completion, Right/Wrong Answer)
- Hint
- Graphics (Display of hangman and its parts)
- Health & Regen (Visual representation of lives using heart shape)
- Selection of questions for each module
- Settings
- Rules

System Overview

The whole game was designed in Visual Studio Code (using Python program with its respective libraries), where all of us group members have been able to collaborate and complete the project via GitHub. This project aims to provide users with a clear understanding of the game's flow, including the outcomes when they win, lose, or enter invalid inputs. The game is structured around the core modules students encounter during their first year's first trimester (Introduction to Computing, Mathematics 1, Computer Architecture and Organization, and Programming Fundamentals). This aligns with our goal to both educate and assess the user's knowledge of these subjects. Each module contains three questions about increasing difficulty (easy, medium, hard), beginning with the easiest and progressing to the most challenging. Upon completion of this game, users will be rewarded with a $20 grab voucher which aims to entice students to take part in this game. Further details will be provided as the report progresses.

III. Methods

The implementation used in the project

For our project, we implemented several key libraries and modules. We utilized **Tkinter** to create the graphical user interface, leveraging its **tkfont** module for custom font styling and **messagebox** to display user alerts and messages. To integrate sound effects, we incorporated **Pygame**, which provided a robust framework for audio handling. Additionally, we used the **os** module for file management and path operations. Lastly, **Pillow** (Python Imaging Library) was employed to manage and manipulate images through its **Image** and **ImageTk** modules, enabling the inclusion of high-quality visuals in the game.

Features

- **Difficulty Levels:**
    - 3 levels, Easy/Medium/Hard, Total 3 questions per module
    - For easy difficulty, when the user inputs 3 correct letters, 1 body part is added
    - For medium difficulty, when the user inputs 4 correct letters, 1 body part is added
    - For hard difficulty, when the user inputs 4 correct letters, 1 body part is added. An additional timer of 60 seconds to make it harder.

- **Sound Effects:**
    - When you are on the main page (background.mp3 will be played)
    - When you enter a wrong or correct letter input. (Correct > Correct.mp3 & Wrong > Incorrect.mp3 will be played)
    - When you finish the game (Complete.mp3 will be played)
    - For different levels, there are different music (Easy > EASY.mp3, Medium > Medium.mp3 & Hard > Hard.mp3)

- **Basic game logic:**
    - Input a letter to the corresponding question
    - After 3 wrong letters have been entered, a hint is shown on the user screen for the corresponding question.
    - Once hangman has all 10 body parts, the game ends and the user will go back to the login screen.
    - Code in a manner where the player cannot get more than 10 body parts.
    - Split our content based on modules (For e.g. Intro to Computing, Programming Fundamentals, Mathematics 1, Computer Architecture and Organization)
    - Attain a grab voucher if you win (incentive for them to try)

- **User interface**
    - (Prompt player): design of buttons, background images

- Once one module is completed, “gray out” the button so the player cannot play the level he has already completed.
- Classic hangman of hangman with lines
- One hangman for **ALL** modules. Health bar is also included together with the hangman for a more accurate visual representation of how much life the player has.

Algorithms design

Our team has discussed the first plan flow of the hangman game, and the system diagram for version 1 is shown below in Figure 1:

(See Figure 1 in the Figures and Tables section within the appendix of the report)

In this program, the user starts by entering their name, student ID, degree, and SIT email. The user is then taken to a module selection page, where they must answer five questions based on core modules. For each question, if the student does not pass, a part of a hangman figure is added. If the hangman reaches 10 parts, the user must select the module again. This process is repeated for a total of four core modules.

Once the user passes a module, the program checks whether all modules have been completed. If any modules are remaining, the cycle continues. After passing all core modules, the user is awarded a voucher and a Record of Achievement (ROA).

After extensive planning, team discussions, and interviews with students, we have transformed our initial system diagram into an improved, more user-friendly experience. This process led us to develop version 2, which reflects these enhancements and provides a smoother, more engaging gameplay experience:

(See Figure 2 in the Figures and Tables section within the appendix of the report)

In this version of the hangman game, the user starts by entering their name and SIT email, simplifying the input process to make it more comfortable and less time-consuming. Once logged in, the student is brought to the home page, where they can select a topic from the available core modules, adjust volume settings, use hints, and even enjoy added music to boost motivation. There’s also a “Quit” option for easy exit.

Upon choosing one of the four core modules, the game initializes with a hangman figure, starting with 10 lives. The student then progresses through three questions: Question 1 (easy), Question 2 (medium), and Question 3 (hard). For each incorrect answer, a life is lost, and the hangman image updates accordingly. If lives remain, the student can continue; however, if lives reach zero, they must restart the game and log in again.

Questions 1 and 2 have no time constraints, but for Question 3, the student has 60 seconds to answer. If the timer runs out, a life is deducted. Upon completing all questions across the four core modules, the student is rewarded with a $20 Grab voucher as an incentive to complete the quiz. If any of the core modules still need to be completed, the student must complete them all to earn the voucher.

The aim and objectives are to make the hangman competitive and at the same time, students might find the motivation and put the effort into the quiz.

Technology Implementation

Our project was developed entirely from the ground up in Visual Studio Code, without the use of any pre-existing templates. Throughout the process, we utilized ChatGPT, W3Schools and Stack Overflow to enhance the efficiency and flow of the game. Drawing inspiration from the traditional gameplay of Hangman, we aimed to recreate the essence of the classic game while implementing modern improvements.

GitHub

Our group's GitHub repository
*(See **Figure 3** in the Figures and Tables section within the appendix of the report)*

Pushing/Pulling updated codes to/from the github repository
*(See **Figure 4 & 5** in the Figures and Tables section within the appendix of the report)*

GitHub is a web-based platform designed for version control and collaboration on software development projects. It uses **Git**, a distributed version control system, to manage and track changes in source code, enabling multiple developers to work on a project simultaneously. GitHub allows users to store their code in repositories, which can be either public or private.

By utilizing GitHub, we were able to continuously push updates to a shared repository and pull the latest changes from it. This streamlined collaboration enabled us to work on the code collectively without the need for manual updates or version management, ensuring that all contributors had access to the most current version of the project.

W3Schools:

One example of a function our group learned from W3Schools
*(See **Figure 6** in the Figures and Tables section within the appendix of the report)*

By referencing W3Schools, we were able to learn and apply new techniques to enhance our project's code. This allowed us to improve both the design and functionality of our game. For instance, we gained a deeper understanding of how to use the class function and when it should be implemented, which was crucial for organizing our code. Additionally, we researched ways to make our Hangman game more interactive and, through W3Schools, learned how to implement Tkinter to create a user-friendly interface.

Stack Overflow:

One example of how our group made use of Stack Overflow
*(See **Figure 7** in the Figures and Tables section within the appendix of the report)*

By referring to stack overflow, we were able to utilize and understand how other people code their programs and understand what the code does. For example, on resetting the game, by understanding other people's code, we were able to integrate their ideas into our code. Some of the codes that we have integrated might cause some errors to arise, hence the need for stack overflow, to advise us on improving and mitigating errors in our codes.

Chat GPT:

An image of the group referring to ChatGPT to improve the lines of codes
*(See **Figure 8** in the Figures and Tables section within the appendix of the report)*

We turned to ChatGPT when documentation was lacking or when seeking ways to enhance our code. When exploring the creative aspects of the project, we found ChatGPT to be invaluable, as it significantly contributed to improving our code. A key example is shown in the image below, where we relied on ChatGPT for guidance on uploading and displaying our custom-drawn Hangman images in the project. It was crucial to link these images to the player's remaining lives, and ChatGPT provided a sample code that inspired us to successfully achieve this objective.

IV. Result and Insights

Experimental Evaluation and Result Analysis

1. Not handling input validation
Mistake: Failing to check if the player's input is valid (e.g., if it's a single letter, if it has already

been guessed, or if it's part of the alphabet).
Solution: Always validate input before proceeding. Check if the input is a single letter and ensure the player hasn't guessed it before.

2. Not updating the game state correctly
Mistake: Failing to update the displayed word with correct guesses or incorrectly handling word replacement.
Solution: Use a list to represent the current state of the word being guessed and update it correctly each time the user guesses a correct letter.

3. Not tracking guessed letters properly
Mistake: Forgetting to store guessed letters, which leads to repeated guessing or incorrect win/loss logic.
Solution: Use a list or set to track all guessed letters (both correct and incorrect) and check if a guess has been made before.

4. Incorrect win/loss conditions
Mistake: The game may not recognize when the player has won (guessed all letters) or lost (exceeded max wrong guesses).
Solution: Check if all letters in the word have been guessed and if the number of wrong guesses exceeds a predefined limit.

5. Not managing case sensitivity
Mistake: The game may treat uppercase and lowercase letters as different guesses.
Solution: Normalize all input and the secret word to lowercase (or uppercase) for consistency.

6. Not implementing a "lives" or "attempts" system properly
Mistake: If the number of wrong guesses isn't tracked properly, the game might go on indefinitely or end prematurely.
Solution: Implement a clear counter for wrong guesses and terminate the game when it reaches the maximum number of allowed attempts.

7. Loading and displaying images incorrectly
Mistake: Failing to correctly load and display the hangman images as the game progresses. This might lead to broken images or improper positioning in the game window.
Solution: Ensure that the images are loaded properly and display them at the correct points during the game. If you're using a GUI framework (e.g., Tkinter, Pygame), be sure the image paths are correct and compatible with the framework's image-handling methods.

8. Incorrect image scaling or stretching
Mistake: Images may not be scaled correctly to fit the window, resulting in stretched or poorly displayed hangman figures.
Solution: Use the correct methods to resize images based on the window or screen resolution while keeping the aspect ratio intact.

9. Mismanaging sound effects
Mistake: Sound effects (e.g., for correct/incorrect guesses, game win/loss) might overlap or play too late due to poor handling of audio channels or buffering.
Solution: Preload the sound effects and ensure they are played at the correct times without overlapping.

10. Incorrect or missing sound synchronization
Mistake: Sound effects might play at inappropriate times, such as too early or after the game logic has been completed.
Solution: Ensure sound effects are tied to specific game events like correct or incorrect guesses, or game-over conditions. Trigger sounds right after the relevant event in the game loop.

11. Not handling different file formats for images and sounds
Mistake: Using incompatible file formats for images or sound effects, leading to errors when trying to load them in the game.
Solution: Ensure that all media files (images, sounds) are in formats supported by your chosen framework.

12. Not handling the Login Page input
Mistake: The user may skip or not follow the input of the login page. (i.e. Leave the input as empty, the email did not put their email account).
Solution: Check on the user input, every time they click on login. Name and email cannot be blank, and email is required for the user to input '@'.

The images below show how fields cannot be empty, and email must be valid. (Expected and actual)
*(See **Figure 9 & 10** in the Figures and Tables section within the appendix of the report)*

12. Hints show when the user has 3 failed attempts
Mistake: The user may constantly get hints for every failed attempt, this can annoy and make the game not to be fun.
Solution: For every level (easy, medium, hard) the game reset the hints. For example, if the user fails 3 attempts, then the user will be given a hint. The logic applies to medium and hard levels.

12. The Guess button is still available when the user has completed the word
Mistake: The user can still click on the guess button despite the word having already been guessed. This can cause the user to still lose their lives, and the error of the guess button is not handled properly.
Solution: Disable the guess button if the word has already been guessed.

12. Next button did not show if the word is already guessed
Mistake: The next button did not show up despite the user having already guessed the word.

This causes the user unable to advance to the next level.
Solution: Show the next button, for a word that has been already guessed.

V. Conclusion

In conclusion, this project successfully developed a Hangman game in Python with a functional graphical user interface (GUI), providing a visually engaging and interactive experience for players. By implementing this project, we gained hands-on experience with Python programming, including GUI development, user input handling, and game logic.

For future enhancements, the game could be expanded with additional features such as customizable word lists, more different types of modules and questions, and themes to provide a more personalized experience. Additionally, integrating a scoring system with an online leaderboard could add a competitive element, allowing players to compare their performance. Expanding the GUI with animations could also further enrich the gameplay experience, making it more immersive and enjoyable.

VI. Appendix

Figures and Tables

Figure 1: Initial Flowchart of WannaHang @SIT

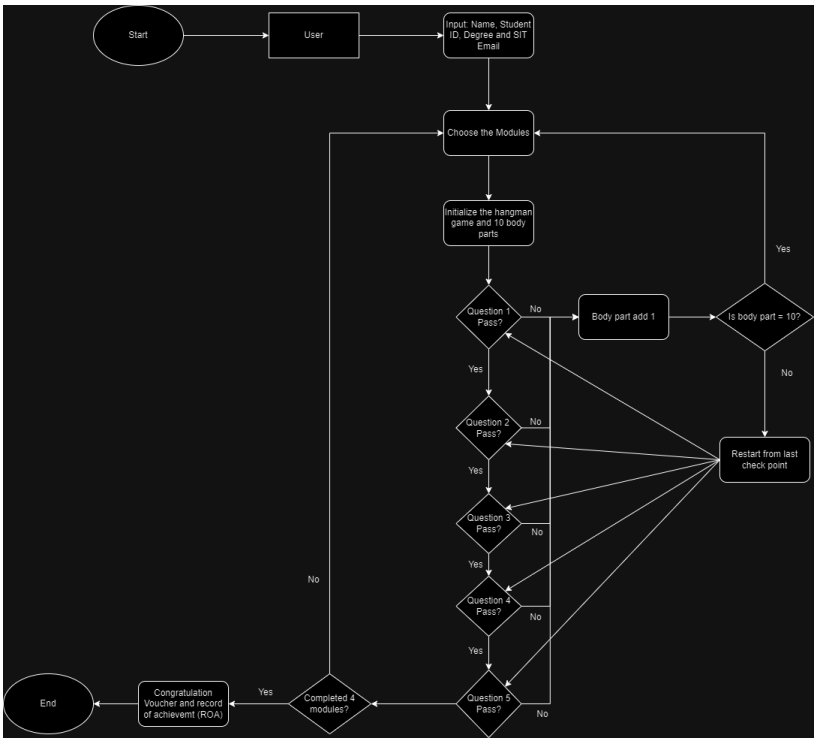


Figure 2: Finalized Flowchart of WannaHang @SIT

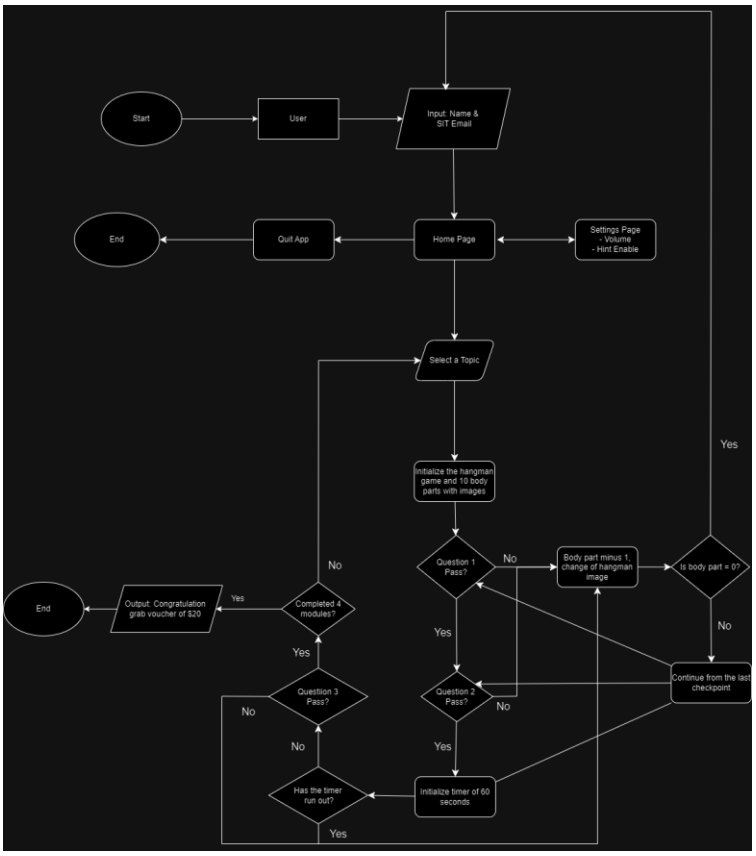


Figure 3: Image of our group's GitHub repository

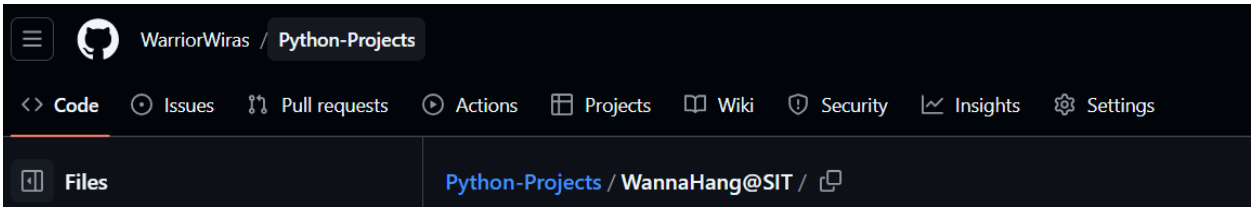


Figure 4 (Push) & 5 (Pull): The process of pushing/pulling our updated files and codes to/from our GitHub repositories

```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

D:\1. SIT Things\1. Class Materials (Uni notes - SIT)\Y1 2024 Tri 1 (ongoing)\INF1002 - Programming Fundamentals\Python
Project\wannahangsit>git add .

D:\1. SIT Things\1. Class Materials (Uni notes - SIT)\Y1 2024 Tri 1 (ongoing)\INF1002 - Programming Fundamentals\Python
Project\wannahangsit>git commit -m "Final Commit"
[main 40e239d] Final Commit
 1 file changed, 2 insertions(+)

D:\1. SIT Things\1. Class Materials (Uni notes - SIT)\Y1 2024 Tri 1 (ongoing)\INF1002 - Programming Fundamentals\Python
Project\wannahangsit>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 324 bytes | 324.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/gemokay/wannahangsit.git
   5f48f32..40e239d  main -> main

D:\1. SIT Things\1. Class Materials (Uni notes - SIT)\Y1 2024 Tri 1 (ongoing)\INF1002 - Programming Fundamentals\Python
Project\wannahangsit>
```

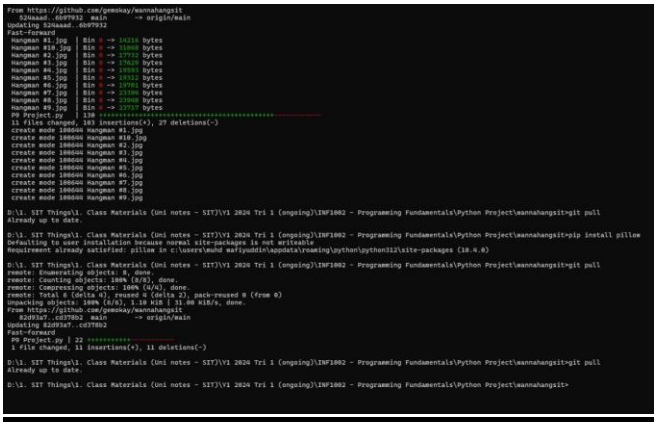


Figure 6: One example of a function our group learned from W3Schools

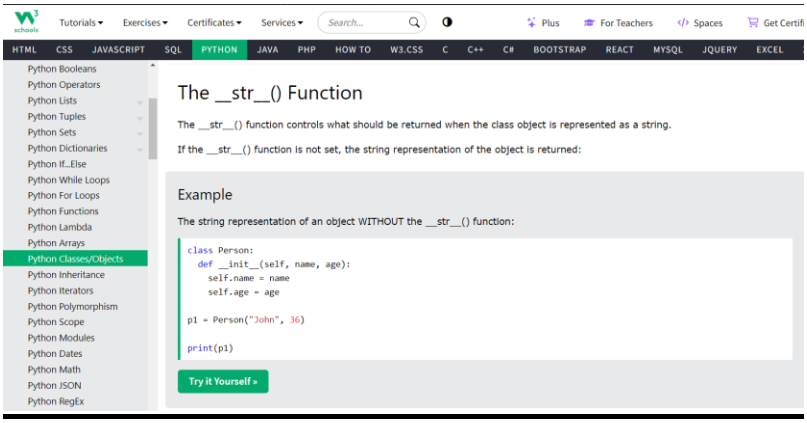


Figure 7: One example of how our group made use of Stack Overflow

Figure 8: Referring to ChatGPT to improve our line of codes

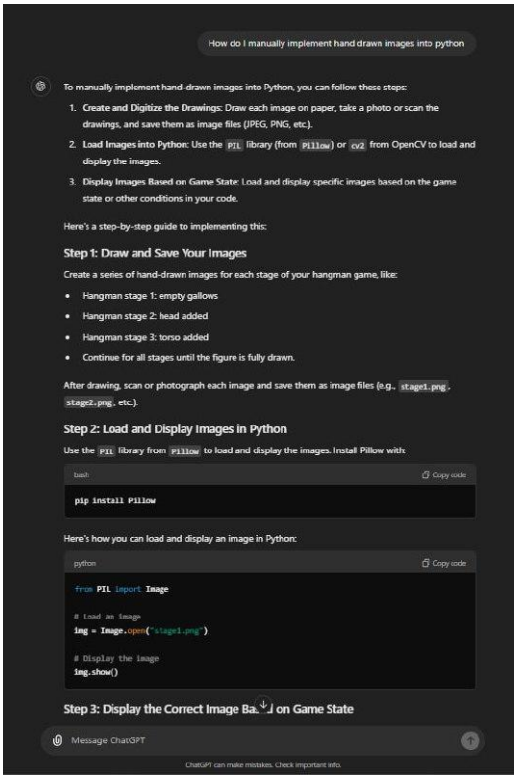
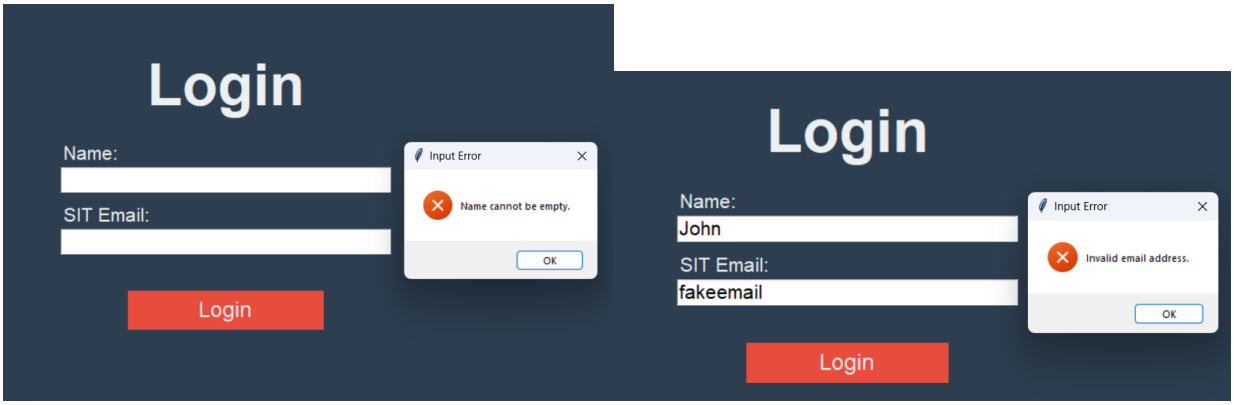


Figure 9 & 10: Fields cannot be empty and email must be valid.(Expected and actual)


Login
Name:
SIT Email:
Login
Input Error
Name cannot be empty.
OK
Login
Name:
John
SIT Email:
fakeemail
Login
Input Error
Invalid email address.
OK