Cohort 27 Python Beginners Project: Hangman

Introduction and Project Overview

This document outlines the requirements and objectives for the Hangman Game project, designed for students who are beginning their study of Python programming. The project entails the development of an interactive word-guessing game, utilizing core programming concepts such as variables, loops, lists, conditional statements, and functions. Hangman is a well-established logic game in which the player attempts to determine a concealed word by proposing individual letters, with a finite number of attempts provided.

Students are expected to follow the instructions provided and implement each specified feature methodically. Successful completion of this project will serve to reinforce fundamental programming skills, encourage structured problem-solving, and enhance understanding of Python syntax and logic. It is recommended that students test their code regularly and maintain a clear and organized approach throughout the development process.

Proceed with the initial stage of construction, keeping in mind that further features will be specified subsequently. The foundation of your Hangman game should be robust, adhering to best practices and demonstrating a thorough comprehension of essential programming principles.

# Startup

When the Hangman program starts, it immediately sets up a guest account for the current user. This guest account acts as a temporary profile that remains active only until the user chooses to log in or register a new account.

* Upon launch, the application prompts the user to enter their name, which is stored in the Guest account for the session.
* This step offers a personalized experience and ensures good practice with user input handling.

The Guest account tracks three session-specific statistics:

* **name**: The name of the Guest user
* **session\_wins**: Number of games won in the current session
* **session\_losses**: Number of games lost in the current session
* **session\_plays**: Total number of games played in the current session

Key points about the Guest account:

* It is strictly temporary; none of its data is saved to disk or made permanent in any way.
* All tracked statistics and the user's name exist only in memory during that session.

If the user decides to log in or register while playing:

* The program transfers the values of **session\_wins**, **session\_losses**, and **session\_plays** from the Guest account to the new or existing user account.
* This ensures the player's progress as a guest is preserved in their permanent profile.

Alongside guest account management, the program also handles persistent storage setup:

* On startup, it checks for a **save** directory in the project folder.
* If the directory does not exist, the program creates it automatically.
* It then ensures an **accounts.json** file exists inside the save directory.
* If **accounts.json** is missing, the program creates an empty file with that name.

This preparation allows for safe and organized management of user accounts in future sessions, while keeping the Guest account completely temporary.

# Features

The Hangman game will be designed to operate entirely on the console, ensuring accessibility for all students regardless of their development environment. Upon launching the program, users will be greeted by a welcoming introductory message that sets the tone for the game experience. Immediately following this, a clear and user-friendly menu will be presented, allowing the user to choose from the following options:

* **Quick Game**: Instantly start a standard round of Hangman with a randomly selected word.
* **Login**: Enter existing credentials to access personalized features or track progress.
* **Register**: Create a new account to save progress or participate in leaderboards (if implemented in future stages).
* **Exit**: Close the program at any time if the user wishes to end their session.

This menu-driven approach encourages clear program structure and demonstrates the application of conditional statements and loops in Python. It also provides a foundation for expanding the game’s features in subsequent stages while supporting robust user interaction from the outset.

## Quick Game Feature

### Immediate Game Start:

The Quick Game mode instantly launches a standard round of Hangman, providing a seamless and fast-paced entry into gameplay for users who want to begin guessing words right away.

* Accessible directly from the main menu
* No need for account setup or login
* Ideal for practice and casual play

### Random Word Generation:

Each game begins with a mystery word chosen at random, utilizing the Python function supplied in the Project Question.

* Ensures variety and unpredictability for every session
* Promotes replayability and broad vocabulary exposure
* Demonstrates use of functions and randomness in Python

### ASCII Art Visualization:

The game uses ASCII graphics to depict the hangman, visually representing the player's progress and remaining chances.

* The hangman drawing updates with every incorrect guess
* Provides a clear, graphical narrative of the game’s stakes
* Enhances engagement and interactivity within the console environment

### Word Display:

The current word is displayed after each guess, showing which letters have been correctly identified.

* Unrevealed letters are masked for suspense
* Correct guesses reveal their positions instantly
* Players can track their progress and strategy visually

### Guess History Tracking:

The program maintains and shows a complete list of all letters guessed—both correct and incorrect.

* Prevents repeated guesses and wasted attempts
* Gives players a quick reference for their strategy
* Encourages careful, logical guessing

### Display of Remaining Lives:

The interface always indicates the number of lives left, keeping players aware of how many incorrect guesses remain before the game ends.

* Acts as a countdown to the game’s conclusion
* Heightens tension and urgency as lives decrease
* Supports the classic rules and progression of Hangman

This hierarchical and itemized feature list ensures that the Quick Game mode is both robust and engaging, fostering a clear understanding of fundamental Python programming concepts while maximizing user experience.

## Login Feature

### Account Directory and File Initialization:

When the program launches, it automatically checks for a designated directory within the project folder dedicated to storing account data.

* If the directory does not exist, it is created on startup.
* Inside this directory, an empty JSON file named accounts.json is initialized (if not already present).

### User Login Process:

* When the user selects the login option, the program prompts the user to enter a username.
* The entered username is checked against the list of usernames stored in accounts.json.
* If the entered username does not match any existing account, the program informs the user that the username was not found and prompts for the username again.
* The user is given up to 5 attempts to enter a valid username.
* Upon a successful username match, the program asks for the associated password.
* The program attempts to find an account in accounts.json where both the username and password match the credentials provided.
* If the password is incorrect, the program notifies the user of the mismatch and prompts for the password again.
* Up to 5 attempts are allowed for the user to enter the correct password.
* If the user fails to enter a correct username or password after 5 attempts, the system presents an option:
* Ask whether the user wishes to continue the login process
* Or exit and return to the previous menu
* This iterative approach enhances user experience by balancing security, usability, and clear guidance.

### Benefits:

* Ensures all account data is managed systematically and securely within the project environment.
* Provides clear, user-friendly prompts and responses at each step.
* Prevents brute-force login attempts by limiting retries and offering logical next steps for the user.
* Reinforces essential Python programming concepts such as file management, iteration, conditional logic, and user interaction.

## Register

### Account Creation:

* Users begin by selecting the “Register” option from the main menu.
* The system prompts the user to enter a desired username. To ensure uniqueness, the entered username is checked against all existing account records within the project environment. If the username is already in use, the system displays a clear, informative message explaining that the username is taken, and asks the user to try again with a different name. This cycle repeats until the user provides a unique username.
* If the user fails to enter a valid username after five (5) attempts, generate a menu asking the user if they wish to continue trying to register or to return to the previous menu and act accordingly.

### Password Entry Choice:

* Once a unique username is chosen, the program asks the user whether they would like to manually enter their password or have one generated automatically.
* If the user chooses manual entry, the system prompts for a password and verifies that the password meets all required security criteria:
* At least one lowercase letter
* At least one uppercase letter
* At least one digit
* At least one symbol
* Maximum length of 16 characters
* If the user fails to enter a valid password after five (5) attempts, generate a menu asking the user if they wish to continue trying to register or to return to the previous menu and act accordingly.
* If the password does not meet these requirements, a clear message is presented, and the user is prompted to try again. This loop continues until a valid password is entered.
* If the user chooses automatic generation, the system creates a password that satisfies all criteria and displays it for confirmation.

### Name Entry:

Prompts the user to enter the name to be called by for the new account being created.

### Process Hierarchy:

* Step 1: Initiate registration via user interface.
* Step 2: Enter and validate username (iterative until unique and valid).
* Step 3: Decide password entry method (manual or automatic).
* Step 4: Enter or generate password, verify against all security requirements (iterative if manual entry).
* Step 5: Enter the name associated with the account
* Step 6: Confirm registration and securely store account data.
* Step 7: Return to main menu.

### Data Management:

* After the user has created a new account, all information concerning the user is stored in a new account which is appended to the **accounts.json** file within the project environment. All file operations are carefully managed to prevent data loss and unauthorised access, ensuring systematic organisation and efficient retrieval.

### Iterative Validation:

* Both username and password entry use iterative prompts and clear feedback to guide users toward successful registration.

### User Guidance:

* The registration workflow offers descriptive prompts, examples where helpful, and logical navigation based on user decisions.

### Programming Concepts:

* File management: Secure storage and retrieval of user credentials.
* Iteration: Repeated input requests until valid, secure data is provided.
* Conditional logic: Rule-based checks for all registration steps.
* User interaction: Responsive prompts and feedback throughout the process.

For each user account, the following fields are stored to provide a detailed record of both overall achievements and the current session's activity:

* **name**: The user's display name associated with the account.
* **username**: The unique identifier chosen by the user for login and account management.
* **password**: The securely stored authentication credential for the account.
* **wins**: The cumulative number of wins achieved by the user since the account's creation.
* **losses**: The total losses accumulated by the user throughout the account's existence.
* **plays**: The aggregate count of rounds played since the account was established.
* **session\_wins**: The number of wins recorded during the current program session—that is, since the user logged in or registered their account.
* **session\_losses**: The losses incurred during the active session, reset upon each new login or registration.
* **session\_plays**: The number of rounds played during the current session, allowing for analysis of recent activity.

This structured approach ensures each user's progress and activity—both historical and session-based—are systematically tracked, enabling personalised feedback, performance summaries, and a robust foundation for enhancing the user experience.

## Exit Feature

### Add Session Results to Account:

When you leave the program, the wins, losses, and total games played for each logged-in user are updated. The program takes the numbers you earned this session—**session\_wins**, **session\_losses**, and **session\_plays**—and adds them to your main totals: **wins**, **losses**, and **plays**.

### Keep Records Up to Date:

This makes sure your progress from this session is saved and your account information stays accurate.

### Save to File:

After updating, the program writes your new totals to a file called **accounts.json** so nothing gets lost when the program closes.

### If You Are a Guest:

If you’re using a guest account, the program does not save anything and just closes. No information from guests is stored.

This process helps regular users keep track of their progress every time they play and lets guests play freely without saving any data.

# Post Login/Register Menu

How the Main Menu Adapts for Logged-In Users

Once a user has successfully logged in or registered for a new account, the main menu transforms to provide options suited to an authenticated experience. The default options of **Quick Game**, **Login**, **Register**, and **Exit** are replaced with the following:

* **Play Game**
* **Logout**
* **Exit**

## Play Game

The Play Game feature allows logged-in users to enjoy matches in the same way as the original *Quick Game* function. Gameplay behaviour and flow remain unchanged; however, session results—such as wins, losses, and number of games played—are now tracked and attributed directly to the logged-in user's account. This ensures users' progress is accurately recorded and updated for every session.

## Logout

The Logout option enables users to securely disconnect from their account without terminating the program. When selected:

* The program updates the user's account details by adding session results—session\_wins, session\_losses, and session\_plays—to their main totals for wins, losses, and plays.
* The new totals are immediately saved to **accounts.json**, preserving all progress and account data before ending the session.
* After saving, the user's account is closed, and the program automatically switches to a fresh guest account, which begins with empty progress and does not store any data.
* The main menu reverts to its original state, offering **Quick Game**, **Login**, **Register**, and **Exit**—just as it appeared before authentication.

This allows users to switch seamlessly between registered accounts and guest play, maintaining data integrity and a smooth user experience.

## Exit

The Exit option continues to operate as before:

* If the user is logged in, all session results are merged with their account totals and saved to **accounts.json** before the program closes.
* For guest users, no data is saved; the program simply terminates with no record of guest activity.

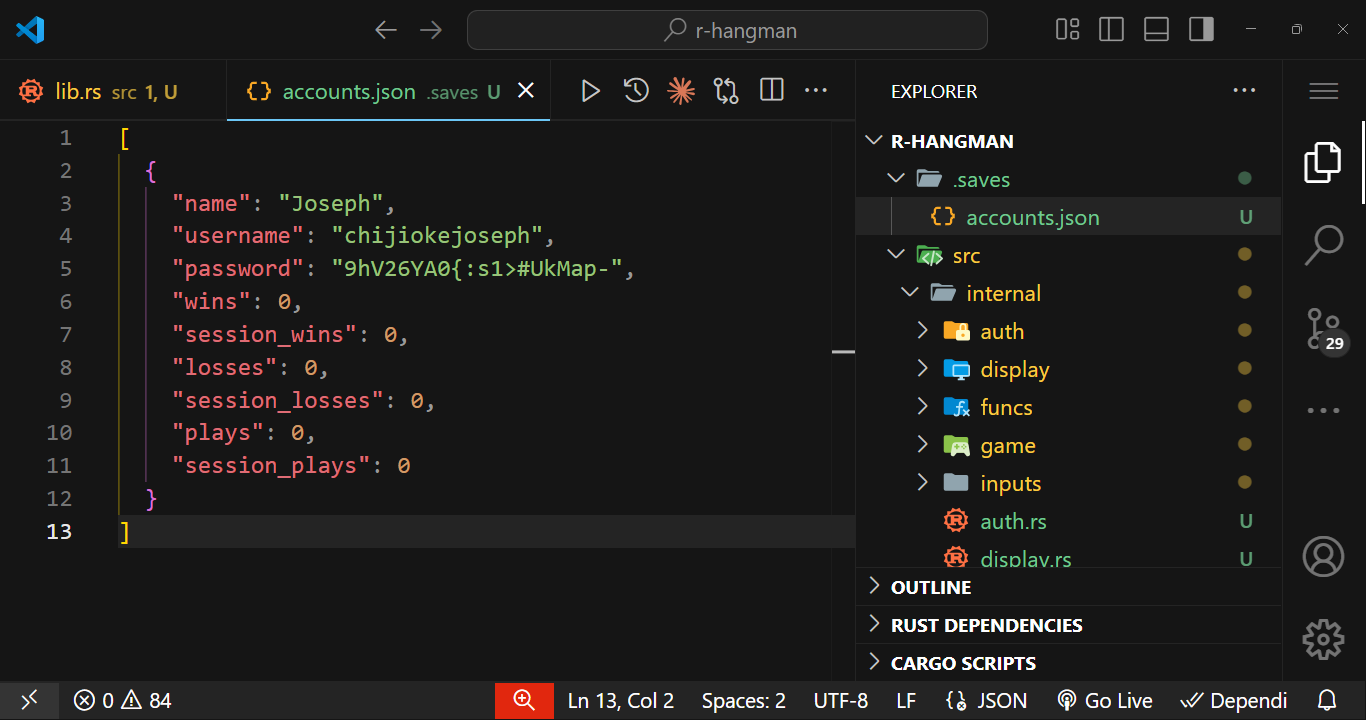
# JSON Structure

The **accounts.json** file contains all registered user data, securely managing accounts and tracking progress. Each account object is indexed by a unique username or ID and includes fields such as:

* **name**: User's display name
* **username**: Account identifier
* **password**: User's password
* **wins**: Total victories
* **losses**: Games lost
* **plays**: Games played
* **session\_wins**: Victories earned in the current session
* **session\_losses**: Losses in the current session
* **session\_plays**: Games played in the current session

At the end of each session, the values for *session\_wins*, *session\_losses*, and *session\_plays* are updated and stored alongside other account details. Guest user activity continues not to be recorded in this file.

A sample of how the JSON file should appear.



# Summary

After login or registration, the main menu shifts from **Quick Game, Login, Register, Exit** to **Play Game, Logout, Exit**. **Play Game** provides classic gameplay with progress tracking, **Logout** saves user data and transitions to a guest profile with the initial menu, and **Exit** reliably saves or discards session data depending on account status.

# Note

Please be reminded of the following important details for the project submission and presentation:

* Each group must submit their completed project codes on or before **Monday, 18th August at exactly 6:00am**.
* A submission link will be provided by **Sunday, 17th August at 6:00pm**. Please ensure you check for access and upload your files promptly.
* **Each group is required to nominate three representatives** for the presentation. This team must include at least one male and one female. **Group leaders and assistant group leaders are excluded from serving as representatives**.
* The presentation date will take place on any day between **Monday, 18th August and Wednesday, 20th August**. The exact date will be communicated once confirmed.
* The preferred presentation mode is **online**. Any changes to the arrangement will be shared in advance.
* **No presentation slides are required** for this project. Please prepare to demonstrate and discuss your code directly.

Thank you for your cooperation and attention to these instructions. All updates or further changes will be communicated as necessary.

# Hint: Python Packages

Here are targeted suggestions for Python libraries and modules that will be especially helpful for building a Hangman game with account management and JSON-based data storage:

* **json** – Use this module to read and write account and progress data to a *.json* file. This makes it easy to load user profiles at login and save progress, such as games played, high scores, or unlocked achievements.
* **requests** – If your Hangman game will have any online features (such as checking for updates, submitting results, or retrieving new word lists), **requests** allows you to interact with web APIs smoothly.
* **random** – Essential for randomly selecting the word to guess from your word list, ensuring each game provides a fresh challenge.
* **os** – Useful for checking if your account data file exists, creating directories, or handling file paths in a cross-platform way.
* **pathlib** – Makes working with file paths and directories more intuitive, especially when managing your JSON data file or storing logs of gameplay history.

For your specific requirements:

* Store user registration and game progress in a JSON file using **json**.
* Randomly select Hangman words with **random**.
* Use **os** and **pathlib** to manage and verify the existence of the data file and handle file paths securely.
* If online functionality is needed (such as fetching word lists), leverage **requests**.

These modules provide a solid foundation for developing your Hangman game, managing account data efficiently, and extending your project with online features if desired.