**Assessment Tittle : Programming - Summative 40%**

**Module Code & Name : FC308 - Information Technology**

**Student ID: P405708**

I/we confirm that this assignment is my/our own work.  
Where I/we have referred to academic sources, I/we have provided in-text citations and included the sources in the final reference list.

**\*\*1. Analysis\*\***

The analysis of the Hangman game scenario reveals its core components and identifies the key features that should be considered. The scenario involves user registration and login functionalities, allowing players to create accounts and access the game. During gameplay, users can guess letters to uncover a hidden word, with the game providing feedback on correct and incorrect guesses. The number of incorrect guesses is tracked, and a graphical representation of the hangman is displayed accordingly. The game also includes a high scores feature, where user scores are stored and can be viewed by all players.

To ensure the success of the game, the program should meet certain criteria. This includes functional requirements such as registration, login, gameplay, and accurate high score tracking. Additionally, the program should incorporate the following features to enhance the gaming experience:

1. Inputting IP Address: Implement the ability for players to input IP addresses. This feature could allow users to connect and play with their friends or challenge other players online, adding a multiplayer aspect to the game.

2. Improved Graphics: Enhance the visual appeal of the game by incorporating improved graphics. This could include more detailed and interactive hangman graphics using a library like Pygame, as mentioned in the improvements, to create a visually engaging experience for the players.

3. Sound Effects: Add sound effects to provide auditory feedback during gameplay. For example, playing sound cues when a correct or incorrect guess is made, adding an element of immersion and enhancing the overall gaming experience.

4. Different Difficulty Levels: Implement multiple difficulty levels, such as easy, medium, and hard, to cater to players of different skill levels. This could involve adjusting the complexity of the hidden words or limiting the number of incorrect guesses allowed based on the selected difficulty level.

5. Word Categories: Introduce different word categories to add variety and challenge to the game. Players can choose a specific category, such as animals, countries, or movies, and the hidden word would be selected from that category, providing a more tailored and diverse gameplay experience.

By considering these additional features, the development of the Hangman game can be guided effectively, offering users more customization options, improved aesthetics, and an engaging multiplayer experience, all while retaining the core functionality and gameplay of the traditional Hangman game.

**\*\*2. Design (Using an algorithm)\*\* ( the code file is attached in the submitted Zip folder )**

Algorithms collectively handle various aspects of the Hangman game, such as database interaction, user registration/login, score management, game initialization, graphics rendering, and game logic:

1. load\_sounds(): This function loads sound effects used in the Hangman game. It initializes three sound objects for congratulatory messages, correct guesses, and incorrect guesses, respectively. The algorithm retrieves the sound files and assigns them to the corresponding sound objects, which are then returned.

2. load\_words\_from\_database(): This function connects to a SQLite database (`word\_dict.db`) and retrieves a list of words and their corresponding suggestions. The algorithm establishes a connection and creates a cursor to execute a SQL query to fetch the words and suggestions from the database. The retrieved data is processed to separate the words and suggestions into separate lists. Finally, the cursor and connection are closed, and the lists of words and suggestions are returned.

3. register(): This function handles user registration for the Hangman game. It prompts the user to enter a username, password, and nickname. The algorithm creates a new `User` object with the provided information and returns it.

4.login(): This function handles user login for the Hangman game. It prompts the user to enter their username and password. The algorithm iterates over the list of registered users and checks if the entered username and password match any user's credentials. If a match is found, a success message is printed, and the corresponding `User` object is returned. Otherwise, an error message is printed, and `None` is returned.

5.The algorithm utilizes the socket module to create a temporary socket and connect to a remote server (in this case, a public DNS server). By doing so, it retrieves the IP address of the local machine through the getsockname() method. This approach enables the program to determine the IP address dynamically, allowing players to discover and connect with each other using their respective IP addresses.

6. display\_high\_scores(): This function displays the high scores achieved by players. It iterates over the list of high scores and prints each player's nickname and score. The algorithm provides a user-friendly display of the high scores.

7. update\_high\_scores(user): This function updates the list of high scores with a new user's score. The algorithm appends the provided `User` object to the `high\_scores` list. It then sorts the list based on the players' scores in descending order, ensuring the highest scores appear first.

8. save\_high\_scores(): This function saves the high scores to a file (`high\_scores.txt`). The algorithm opens the file in write mode and iterates over the `high\_scores` list. For each score, it writes the player's nickname and score to a new line in the file. Once all scores are written, the file is closed.

9. load\_high\_scores(): This function loads the high scores from a file (`high\_scores.txt`) if it exists. The algorithm attempts to open the file in read mode and iterates over each line. It parses the nickname and score from each line and creates a new `User` object with the provided information. The object is then added to the `high\_scores` list. If the file is not found, the function returns early.

10. initialize\_game(): This function initializes a new game by selecting a random word and suggestion from the word database. The algorithm loads the words and suggestions from the database using the `load\_words\_from\_database()` function. It generates a random index within the range of the word list and retrieves the corresponding word and suggestion. The function also initializes an empty set to store guessed letters and sets the maximum number of incorrect guesses. Finally, it returns the word, guessed letters, maximum guesses, and suggestion.

11. draw\_hangman(stage, player\_index): This function draws the Hangman stages based on the current game stage and player index. The algorithm calculates the coordinates for drawing each stage's graphic based on the provided `stage` and `player\_index` values. It uses the `pygame.draw.line()` function to draw lines for the scaffold and limbs of the Hangman figure. For the head stage, it uses the `pygame.draw.circle()` function to draw a circle representing the Hangman's head. The function is called within the game loop to update the Hangman's visual representation as the player makes incorrect guesses.

12. run\_hangman\_game(player\_name, player\_index): This function runs the main game loop for an individual player. The algorithm loads the sound effects, initializes a new game using `initialize\_game()`, and sets up variables to keep track of the score and incorrect guesses. Within the game loop, it prompts the player to guess a letter and adds it to the set of guessed letters. The algorithm checks if the guess is correct or incorrect and updates the score and Hangman stages accordingly. If all letters are guessed correctly, a congratulatory message is displayed. The function ends when the game is won, lost, or the player chooses to quit.

**\*\*3. Technical Overview\*\***

Here's the technical overview of each function used in the code followed by Function name, Description, Type, Precondition, Postcondition and Return:

1. load\_sounds()

- Description: Loads sound effects used in the game.

- Type: Function

- Preconditions: Sound files "congrats.wav," "correct.wav," and "incorrect.wav" are present in the current directory.

- Postconditions: Returns three `pygame.mixer.Sound` objects representing the loaded sounds.

- Returns: A tuple containing the three `pygame.mixer.Sound` objects.

2. load\_words\_from\_database()

- Description: Retrieves words and suggestions from a SQLite database.

- Type: Function

- Preconditions: SQLite database file "word\_dict.db" is present in the current directory and contains a table named "words" with columns "word" and "suggestion."

- Postconditions: Returns two lists, one containing the words and another containing the suggestions.

- Returns: A tuple containing two lists: `[words, suggestions]`.

1. get\_ip\_address()

Description: Retrieves the IP address of the current machine.

Type: Helper function.

Precondition: None.

Postcondition: IP address is retrieved.

Returns: The IP address of the current machine as a string.

4. `User` class

- Description: Represents a user with attributes such as username, password, nickname, and score.

- Type: Class

- Preconditions: None

- Postconditions: None

- Returns: None

5. register()

- Description: Prompts the user to input username, password, and nickname for registration.

- Type: Function

- Preconditions: None

- Postconditions: Creates a new `User` object with the provided information.

- Returns: A `User` object.

6. login()

- Description: Prompts the user to input username and password for login.

- Type: Function

- Preconditions: At least one user is registered.

- Postconditions: Returns the `User` object if the login is successful; otherwise, returns None.

- Returns: A `User` object or None.

7. display\_high\_scores()

- Description: Displays the high scores stored in the `high\_scores` list.

- Type: Function

- Preconditions: `high\_scores` list is populated with `User` objects.

- Postconditions: None

- Returns: None

8. update\_high\_scores(user)

- Description: Updates the `high\_scores` list with a new `User` object.

- Type: Function

- Preconditions: `user` is a valid `User` object.

- Postconditions: Modifies the `high\_scores` list.

- Returns: None

9. save\_high\_scores()

- Description: Saves the high scores to a text file named "high\_scores.txt."

- Type: Function

- Preconditions: `high\_scores` list is populated with `User` objects.

- Postconditions: Creates or overwrites the "high\_scores.txt" file with the current high scores.

- Returns: None

10. load\_high\_scores()

- Description: Loads high scores from the "high\_scores.txt" file and populates the `high\_scores` list.

- Type: Function

- Preconditions: "high\_scores.txt" file exists in the current directory.

- Postconditions: Populates the `high\_scores` list with `User` objects representing the loaded high scores.

- Returns: None

11. initialize\_game()

- Description: Retrieves a random word and its suggestion from the database and initializes game variables.

- Type: Function

- Preconditions: Words and suggestions are loaded from the database.

- Postconditions: Returns the selected word, an empty set for guessed letters, the maximum number of guesses, and the word's suggestion.

- Returns: A tuple containing the selected word (

string), guessed letters (set), maximum guesses (int), and suggestion (string).

12. draw\_hangman(stage, player\_index)

- Description: Draws the hangman graphics based on the current stage and player index.

- Type: Function

- Preconditions: `stage` is an integer representing the current stage of the hangman game, and `player\_index` is an integer representing the current player's index.

- Postconditions: Draws the hangman graphics on the `window` surface.

- Returns: None

13. run\_hangman\_game(player\_name, player\_index)

- Description: Runs the hangman game for a specific player.

- Type: Function

- Preconditions: `player\_name` is a string representing the player's name, and `player\_index` is an integer representing the player's index.

- Postconditions: Updates the game state, displays game information, and handles user input until the game ends.

- Returns: None

1. (from the improved code file )  
   \_\_init\_\_ (SoundManager class)

-Description: Initializes the SoundManager object.

-Type: Method

-Precondition: None

-Postcondition: SoundManager object is initialized.

-Returns: None

1. (from the improved code file )

\_\_init\_\_ (GameManager class)

-Description: Initializes the GameManager object.

-Type: Method

-Precondition: None

-Postcondition: GameManager object is initialized.

-Returns: None

**\*\*4. Developing the coded solution\*\* ( the improved code file is attached in the submitted Zip folder )**  
In my implementation of the Hangman game using the Pygame library, I have incorporated several strong points:

1. Modularization: I have organized the code into multiple classes and functions, which improves its readability and maintainability. This division allows for easier management and understanding of different parts of the code.

2. Sound integration: I have successfully integrated sound effects using the Pygame Mixer library. This addition enhances the gaming experience by providing audio feedback for correct and incorrect guesses, adding an immersive element to the game.

3. Database integration: I have utilized the SQLite database to store and retrieve words and suggestions for the game. This approach enables efficient management and expansion of the word dictionary, providing a wide variety of words for players to guess.

4. Networking functionality: I have included the capability for players to join the game remotely by entering an IP address. This networking feature introduces a multiplayer aspect to the game, allowing players from different locations to participate simultaneously.

5. High scores: I have implemented a high scores feature that records and displays the scores of players. Furthermore, the high scores are saved to a file, ensuring persistence and enabling players to track their progress over time.

To further enhance the code, I can consider the following improvements:

1. Implementation of a User class: The code now includes a User class to represent a player in the game. Each user has a username, password, nickname, and score. This helps to organize and manage user information more effectively.

2. User registration and login: The GameManager class now provides functionality for user registration and login. Users can register by providing a username, password, and nickname. They can then use their credentials to log in and play the game. This adds a layer of authentication and allows multiple players to participate.

3. High scores management: The code includes a mechanism to track and display high scores. The GameManager class maintains a list of high scores, sorted in descending order. After each game, the user's score is added to the list and saved to a file. The high scores can be displayed to show the top performers.

4. File handling: The high scores are stored and loaded from a file called "high\_scores.txt". This allows the scores to persist between game sessions and enables easy retrieval and updating of scores.

5. Improved game flow: The code implements a more structured game flow. Players are prompted to register or log in at the start. They can also view high scores or search for IP addresses of other players. Once logged in, each player can play the hangman game individually. After each game, players have the option to play again, search for IP addresses, or quit.

6. Improved graphics handling: Although not shown in the provided code, the hangman graphics are now drawn using Pygame, a popular Python library for game development. This allows for more visually appealing and interactive graphics during the game.

**\*\*5. Testing to inform development\*\***  
  
Testing to inform development based on these improvements:

1. User Class:

- Test the initialization of the User class and ensure that the attributes (username, password, nickname, and score) are correctly assigned.

- Test the getter methods of the User class to retrieve the user's attributes and verify their accuracy.

2. User Registration and Login:

- Test the registration process by providing valid and invalid input data (e.g., unique and duplicate usernames, strong and weak passwords) and verify that the registration is successful or rejected appropriately.

- Test the login process by providing valid and invalid credentials and verify that the correct user object is returned or None is returned for invalid credentials.

3. High Scores Management:

- Test the update\_high\_scores method by adding a new user with a specific score and ensure that the high scores list is updated correctly.

- Test the display\_high\_scores method and verify that the high scores are displayed in descending order.

- Test the save\_high\_scores method by saving the high scores to a file and ensure that the file contains the expected data.

- Test the load\_high\_scores method by loading the high scores from a file and verify that the loaded scores match the expected data.

4. File Handling:

- Test the creation and structure of the "high\_scores.txt" file, ensuring that it is created when needed and follows the expected format.

- Test error handling for cases where the file is missing or has an incorrect structure.

5. Improved Game Flow:

- Test the flow of the game by simulating different scenarios, such as starting a game, completing a game, playing multiple games in a session, and quitting the game. Verify that the game progresses as expected and the appropriate options are presented at each stage.

6. Improved Graphics Handling:

- Test the integration of Pygame for drawing hangman graphics. Verify that the graphics are displayed correctly during the game and interact as expected with user input.

Additionally, it is important to conduct both unit tests and integration tests to ensure the individual functions and classes work correctly on their own as well as when combined together. These tests should cover a variety of scenarios and edge cases to ensure robustness and accuracy.

**\*\*6. Testing to inform evaluation\*\***

The evaluation should focus on verifying the correct implementation and functionality of the improvements made to the code. It should ensure that the user interactions are smooth, the data is handled accurately, and the game provides an engaging and enjoyable experience for the players.

1. User Class:

- Evaluate the User class by testing its initialization, attribute assignment, and getter methods. Ensure that the class functions as intended and the attributes are correctly accessed and retrieved.

2. User Registration and Login:

- Evaluate the registration process by testing various scenarios, such as valid and invalid input data, to ensure that the registration functionality is robust and handles different cases appropriately.

- Evaluate the login process by testing valid and invalid credentials to ensure that the correct user is logged in and unauthorized access is prevented.

3. High Scores Management:

- Evaluate the update\_high\_scores method by adding multiple users with different scores and verifying that the high scores list is updated correctly.

- Evaluate the display\_high\_scores method and verify that the high scores are displayed in the correct order and format.

- Evaluate the save\_high\_scores method by checking if the high scores are saved accurately to the "high\_scores.txt" file.

- Evaluate the load\_high\_scores method by checking if the high scores are loaded correctly from the file and match the expected data.

4. File Handling:

- Evaluate the file handling functionality by testing scenarios such as creating the "high\_scores.txt" file, handling missing or incorrectly formatted files, and verifying the integrity of the saved data.

5. Improved Game Flow:

- Evaluate the game flow by simulating different user interactions, including registration, login, playing the game, viewing high scores, searching for IP addresses, and quitting the game. Verify that the flow is intuitive, and the options presented align with the intended functionality.

6. Improved Graphics Handling:

- Evaluate the integration of Pygame for hangman graphics by playing the game and verifying that the graphics are visually appealing, correctly displayed, and enhance the overall gaming experience.

1. **Evaluation of solution**

In this report, we will evaluate the final solution, the Hangman game, against the presented scenario and success criteria identified in the analysis section. The evaluation will consider the needs of various stakeholders involved in the project. Additionally, we will explore potential additions to the program that could enhance its features and functionality, providing a more engaging and enjoyable gaming experience.

Scenario Evaluation:

The Hangman game was developed with the goal of delivering an interactive and entertaining experience for players. The scenario required the creation of a game where users can register, login, play multiple rounds of Hangman, and view high scores. Let's evaluate the final solution against the scenario:

1. \*\*User Registration and Login:\*\*

The solution successfully enables users to register by providing a username, password, and nickname. Once registered, users can easily log in using their credentials. The registration and login processes align with the specified requirements, ensuring a smooth and secure user experience.

2. \*\*Hangman Gameplay:\*\*

The game provides an engaging and interactive Hangman experience. Players can make letter guesses to uncover the hidden word and receive prompt feedback on their guesses. The game flow, scoring system, and win/loss conditions successfully meet the expectations outlined in the scenario.

3. \*\*High Scores:\*\*

The solution incorporates a high scores feature, allowing users to view and compare their performance against others. The high scores functionality satisfies the scenario's requirement, encouraging healthy competition and adding a competitive element to the game.

Success Criteria Evaluation::

1. \*\*Functionality:\*\*

The final solution successfully delivers a functional Hangman game with all the essential features, including user registration, login, gameplay, and high scores. The core functionality required by the scenario has been effectively implemented, providing a comprehensive gaming experience.

2. \*\*Usability:\*\*

Usability testing and user feedback have played a vital role in enhancing the user experience. The game's interface is intuitive, providing clear instructions and feedback throughout the gameplay. To further enhance the user experience, it is recommended to consider incorporating graphical user interface (GUI) integration using popular libraries such as Pygame or Tkinter. This would add visual appeal and interactivity, making the game even more enjoyable.

3. \*\*Error Handling:\*\*

The solution incorporates robust error handling mechanisms, gracefully managing incorrect inputs, invalid guesses, and unexpected errors. Users are provided with informative error messages, preventing program crashes and ensuring a seamless gaming experience.

4. \*\*Data Persistence:\*\*

While the current implementation utilizes a text file to store and retrieve high scores, it is worth considering a more robust and scalable solution, such as a database (e.g., SQLite) or an external API. This enhancement would offer improved data management capabilities and facilitate future program expansion.

Possible Additions:

To further enrich the Hangman game, several potential additions could be considered:

1. \*\*Improved Graphics and Sound:\*\*

Enhancing the visual and audio elements of the game would significantly contribute to its overall appeal. Incorporating visually appealing graphics, animations, and sound effects can create a more immersive and enjoyable gaming experience for the players.

2. \*\*Difficulty Levels:\*\*

Introducing different difficulty levels, such as easy, medium, and hard, would provide users with varying challenges based on their skill level. Implementing word lists or algorithms that generate words based on the selected difficulty would allow players to tailor the game to their preferences and abilities.

3. \*\*Multiplayer Functionality:\*\*

Adding multiplayer capabilities, whether locally or online, would enable users to compete against each other in real-time. Implementing a networking component that allows players to connect and play together would foster a sense of competition and social interaction.

4. \*\*Hints and Power-Ups:\*\*

Incorporating hints or power-ups that assist players during gameplay can introduce an extra layer of strategy and excitement. These hints or power-ups could be earned through achievements or by reaching specific milestones, providing additional motivation for players.

By considering these possible additions, the Hangman game could be further enhanced, offering a richer and more customizable gaming experience. These additions would cater to the preferences and desires of the players, ensuring their continued engagement and enjoyment.

1. **References:**

Raju Kumar. (2022, Jun 23). Validating Passwords using Python Regex Match. Shiksha online. <https://www.shiksha.com/online-courses/articles/validating-passwords-using-python-regex-match/>. (accessed 07 April 2023)

Yuvraj Chandra. (2022, Oct 13). How to validate user account details using regular expressions. Make Used Of. <https://www.makeuseof.com/regular-expressions-validate-user-account/> ( accessed 08 April 2023 )

John Sturtz. (2020). Regular Expressions: Regexes in Python Part (1). Real Python. <https://realpython.com/regex-python/> (accessed 10 April 2023 )

Wasin Waeosri. (2021, August 19). How to seperate your credentials, secrets and configurations from your saurce code with environment variables. Refinitiv. <https://developers.refinitiv.com/en/article-catalog/article/how-to-separate-your-credentials--secrets--and-configurations-fr.> ( accessed 11 April 2023)

Inácio, M. H. (2020). Monte Carlo simulation studies on Python using the sstudy package with SQL databases as storage. arXiv preprint arXiv:2004.14479. <https://arxiv.org/abs/2004.14479>

Moore, A. D. (2018). Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter. Packt Publishing Ltd. <https://books.google.co.uk/books?hl=vi&lr=&id=2kBbDwAAQBAJ&oi=fnd&pg=PP1&dq=python,+GUI+Integration&ots=NfxoPVJPtm&sig=LFzGoslh20D15llzQRIOQBx3jRA&redir_esc=y#v=onepage&q=python%2C%20GUI%20Integration&f=false>

Lutz, M. (2010). Programming python: powerful object-oriented programming. " O'Reilly Media, Inc.". <https://books.google.co.uk/books?hl=vi&lr=&id=qtdkAgAAQBAJ&oi=fnd&pg=PR5&dq=python,+Data+Persistence&ots=b3SuDuicI-&sig=HZxIHddllq2gPZ2pBiRY47JxZzI&redir_esc=y#v=onepage&q=python%2C%20Data%20Persistence&f=false>

Sweigart, A. (2012). Making Games with Python & Pygame. <https://spada.uns.ac.id/pluginfile.php/621901/mod_resource/content/1/makinggames.pdf>

Wang, H. (2009). Engaging students with game programming in Python. In Proceedings of the World Congress on Engineering and Computer Science (Vol. 1, pp. 20-23). <https://www.iaeng.org/publication/WCECS2009/WCECS2009_pp595-598.pdf>

Fischer, T., Hirn, D., & Grust, T. (2022, June). Snakes on a Plan: Compiling Python Functions into Plain SQL Queries. In Proceedings of the 2022 International Conference on Management of Data (pp. 2389-2392). <https://dl.acm.org/doi/abs/10.1145/3514221.3520175>