**Game Project**

**By:**

**Kolby Boyd**

**&**

**Princewill Nwokeke**

**Pseudocode: World Adventure Quest**

1. Create a void function ‘titleScreen’ to display the game’s title and rules.
   1. Open the file ‘Title.txt’
   2. Open the file ‘Rules.txt’
   3. Print the title
   4. Print the rules
2. Create a non-void function ‘endgame’ for a display at the end of the game.
   1. Print the title
   2. Print congratulatory message
   3. Print the total correct answers out of 15
   4. Define percentage\_score as a ratio of total correct answers to the total questions
   5. Print the percentage score with percentage sign
   6. Thank the player for playing
3. Create a void function ‘separator’ to draw a row of dashes (“-”) to separate sections of the game.
   1. Print 75 dashes in a row each time this function is called
4. Create a function ‘readStories’ to open and read the ‘Stories.txt’ file.
   1. Initialize an empty dictionary to store each level’s story
   2. Define current level as ‘None’
   3. Initialize ‘current\_story’ as an empty list to store the lines of the current story from the file
   4. Open the file
      1. Read the lines from the file and store them in a list called ‘lines’
   5. Iterate through each line in the list of lines read from the file
      1. Remove leading and trailing whitespace
      2. Check if the current line begins with string ‘Level’
         1. Check if current\_level has been assigned a value other than None
            1. Join the lines of the current story into a single string then assign to the corresponding level in the ‘story\_per\_level’ dictionary
            2. Reset ‘current\_story’ to an empty list, ready for the next story
         2. Extract the level number from the current line, convert it to an integer, and assign to ‘current\_level’
      3. If the line does not start with ‘Level’, it is part of the story content.
         1. Append the current line to the list of lines for the current story
   6. After the loop, check if there was a current level being processed. If so, this means that there is a story associated with it that hasn’t been added to ‘story\_per\_level’ yet
      1. Join the remaining lines of the last story into a single string and add to ‘story\_per\_level’
   7. Return the dictionary containing the stories for each level
5. Create a function ‘singleQuestion’ with two parameters (‘question’ represents the question to be presented to the player and ‘choices’ is a list of answer choices for them to select from)
   1. Print the question to the console, with a newline character for better formatting
   2. Iterate over the list of choices using the ‘enumerate’ function to provide the index, ‘i’, and corresponding choice. This starts from 1.
      1. Print each choice along with corresponding number in the console
   3. Start an infinite loop, prompting the player for input until a valid choice is made or the program is terminated
      1. Prompt the player to input their answer (strip whitespace)
      2. Check if the player provided any input
         1. Start a try-except block to handle potential errors during conversion of user input to an integer
            1. Convert user input to an integer and subtract 1 to align with zero-based indexing
            2. Check if the entered index is within range of valid choices

Return the selected choices corresponding to index provided by user

* + - * 1. Print an error message if the entered index is not within the valid choices
      1. Catch the valueError exception raised if the user input cannot be converted to int.
    1. Print an error message if the user input is not a valid integer
  1. If the player did not provide any input (empty string), prompt them to enter a valid choice.

1. Create a function ‘readQuestions’ to take the ‘filename’ parameter which represents the name of the file
   1. Open the file in read mode and be sure to close it even if an exception is raised
      1. Read all the lines from the file and store in a list called ‘lines’
   2. Initialize an empty dictionary to store questions grouped by level
   3. Initialize a variable ‘current\_level’ to keep track of the current level being processed. Starts at None
   4. Initialize an empty list to store questions for the current level
   5. Iterate over each line in the file
      1. Remove whitespace
      2. Check if the line starts with ‘Level’
         1. Check if ‘current\_level’ has been assigned a value
            1. Associate the current questions list with the current level in the ‘questions\_per\_level’ dictionary
            2. Reset the current list of questions to an empty list
         2. Extract the level number from the line, convert to integer and assign to ‘current\_level’
      3. If a line does not start with ‘Level’, assume it is a question
         1. Split the line into arts using ‘, ‘ as a delimiter and store them in a list called ‘parts’
         2. Extract the question from the first part of the line
         3. Extract the choices from the remaining parts of the line
         4. Append a tuple containing the question and answer choices to the list of current questions
   6. Check if ‘current\_level’ has been assigned a value. This is to make sure that the last set of questions is included in ‘questions\_per\_level’
      1. Associate the final list of questions with the last level in the ‘questions\_per\_level’ dictionary
   7. Return the dictionary with the questions grouped by their level
2. Create a similar function ‘readAnswers’ to that the filename parameter as well
   1. Initialize an empty dictionary to store correct answers, with the level numbers as keys
   2. Open in read mode and close it even if an exception is raised
      1. Read all the lines from the file and store in a list called ‘lines’
   3. Iterate over each line from the file
      1. Remove whitespace, split using ‘:’ as a delimiter and store in a list called ‘parts’
      2. Split the first part of the line using ‘ ‘ (whitespace) as the delimiter, extract the level number, convert to integer, and assign to variable ‘level’
      3. Strip leading and trailing whitespace from the second part of the line (which contains the answers), split the answers into individual choices using ‘, ‘ as the delineate and store in a list called ‘answers’
      4. Associate the list of correct answers to the level number in the ‘correct\_answers’ dictionary.
   4. Return the dictionary with all the correct answers for each level with the level number as the key.
3. Create the main loop void function ‘main’
   1. Start an infinite loop
      1. Call the ‘titleScreen’ function to retrieve the title and rules from ‘Title.txt’ and ‘Rules.txt’ and assign the returned values the the variables ‘title’ and ‘rules’
      2. Call the separator function to print a line of dashes. This visually separates sections of output
      3. Ask the player if they would like to play and store their response. Use .lower to make the input case insensitive
      4. Check if the player’s answer is not yes
         1. If not, print “Bye bye!” and exit the function using return, ending the game
      5. Initialize various variables’ starting values
         1. Set lives to 3
         2. Set current level to 1
         3. Set total correct answers to 0
         4. Set total questions to 0
      6. Load the correct answers for each level from ‘Answers.txt’
      7. Load the correct questions and answer choices for each level from ‘Questions.txt’
      8. Load the correct story for each level from ‘Stories.txt’
      9. Make sure that the game will continue as long as the player has lives left and the current level is less than or equal to five.
         1. Print a line of dashes to visually separate the output
         2. Retrieve and print the story for the current level from the ‘stories\_per\_level’ dictionary
         3. Print a header indicating the questions for the current level
         4. Set the ‘correct\_answers\_count’ to 0
         5. Set the ‘incorrect\_answers\_count’ to 0
         6. Iterate over each question in the ‘questions’ list for the current level
            1. Call the ‘singleQuestion’ function to present the question and choices to the player and store their answer
            2. Compare the player’s answer to the correct answer and update the ‘correct\_answers\_count’ if correct

Print ‘Correct!’

* + - * 1. If player’s answer is incorrect

Update ‘incorrect\_answers\_count’

Remove a life from ‘lives’

If the lives are less than or equal to 0

Break the loop, ending the game

* + - * 1. Call the separator function
      1. Increment the total number of correct answers by the number of correct answers for the current level.
      2. Add the total number of questions for the current level to the overall count
      3. Calculate the total number of incorrect answers
      4. If the player runs out of lives
         1. The game ends and ‘Game over!’ message prints
         2. Break the loop and exit the game
      5. If the player has answered at least two questions correctly
         1. they pass the level
         2. If the current level is 5

Break the loop to end the game

* + - * 1. Otherwise, progress to the next level
        2. If the player is progressing, they are prompted to press Enter to continue
        3. If the player has fewer than 3 lives, they gain a life
        4. If the player answers less than two questions correctly, they fail the level

Print ‘You failed’ message

Break the loop, end the game

* + 1. Call the separator function to visually separate the endgame function from the rest of the output
    2. Print the endgame message, displaying the player’s final score, and thank them for playing
    3. Call the separator again
    4. Prompt the player to play again (yes/no)
    5. If they choose not to play again
       1. Print “Bye bye!”
       2. Break the loop to end the game
       3. If they want to play again, the loop continues and restarts the game

1. Ensure that the main function is executed iff the script is run directly, not when imported as a module into another script.