```
-------------------------------------------------------------------
|By submitting this assignment, we agree to the following:        |
|Aggies do not lie, cheat, or steal, nor tolerate those who do.   |
|We have not given or received any unauthorized aid on this       |
|assignment.                                                      |
|Names: Peter, Quinton, Jared, Anish                              |
|Section: ENGR 102 536                                            |
|Assignment: Lab7Team_Plan                                        |
|Date: 10/24/23                                                   |
-------------------------------------------------------------------
```

Comparing Array Values

- **Variable list**
  - num_widgets (list) (user input)
  - num_days (list) (length of num_widgets)
  - jump_length (list) (values ranging from 1 to (length of num_widgets - 1). Represents the intervals between days to compare)
  - differences (list) (difference between each day)
- **Logic**
  - Create a list for the # of widgets
    - Add user input to the list until a negative number I entered by using a While loop
    - Terminate the loop when the user inputs a negative number
  - Determine the Number of Days
    - Calculate the length of the num_widgets list and store it in the num_days list
  - Generate a jump_lengths list
    - Using a for loop, add values from 1 up to the length of num_widgets minus one to the jump_lengths list. This represents the intervals between days for which we will compare widget numbers
  - Calculate the Differences for Each Jump Length
    - Initialize the differences list.
    - Using a nested for loop:
      - The outer loop iterates over each value in jump_lengths
      - The inner loop iterates over the num_widgets list and calculates the difference in widgets made between two days separated by the current jump length.
  - Display the Differences

        - For each difference:
            - Count how many differences are positive (indicating an increase in widgets made) and how many are negative (indicating a decrease).
            - Calculate the percentage of days that experienced an increase or decrease in the number of widgets made.
            - Print these percentages alongside the associated jump length.

Making the Cut in Golf

- **Variable list**
    - Player_score (list) (holds the scores and names)
    - First_round_scores, second_round_scores (list) (hold the scores of players in the respective rounds)
    - names (list) (hold the names of the players)
    - Sorted_round_1, sorted_round_2 (list) (Sorted lists of the scores from the first and second rounds)
    - median_1, median_2 (The median scores of the first and second)
    - safe_scores_1, safe_scores_2 (list)(holds scores below the median for the first and second rounds)
    - Safe_names, cut_names (list) (players who are safe or unsafe)
- **Logic**
    - **Collect Player Scores and Name**
        - Using a `while` loop, continuously prompt the user for scores of two rounds and the player's name.
        - Append the scores and names to the `player_score` list.
        - Terminate the loop when a negative value is input for the first score.
    - **Separate Scores and Names**
        - Iterate through player_score and segregate the scores from both rounds and names into their respective lists.
    - **Sort Scores**
        - Use the sort() function to obtain sorted lists of scores from both rounds.
    - **Calculate Median Scores**

        - If the number of scores is even, the median is the average of the two middle scores.
        - If the number of scores is odd, the median is the middle score.
    - **Identify "Safe" and "Cut" Players**
        - A player is safe if their scores from both rounds are present in safe_scores_1 and safe_scores_2. Populate the safe_names list with such players.
        - Any player not in safe_names is considered cut. Populate the cut_names list with such players.
    - **Output Results**

Chessboard Moves

- **Variable list**
    - Board (list)(A 2D list that represents a chess board, where uppercase letters represent white pieces and lowercase letters represent black pieces)
- **Logic**
    - **Initialize Chess Board**
        - Use the default configuration of the board that is provided. Initialize as a list.
    - **Game Loop**
        - A continuous loop runs, representing the turns of the game
    - **Display Board**
        - At the start of each loop iteration, a separator is printed to show turns, and the current state of the board is printed using the print_board() function.
    - **Choose a Piece to Move**
        - Prompt the user to select a piece to move by entering its position in xy notation
        - This position is parsed to extract the row and column for the piece
        - If the selected position is empty (contains '.'), an error message is shown, and the program ends
    - **Choose Target Position**
        - Prompt the user to enter the target position for the chosen piece, again using xy notation
        - Parse the input to extract the row and column indices for the target position
    - **Move Piece**

- Retrieve the selected piece from the board
- Set its original position to be empty ('.')
- Place the piece in the target position on the board