**Mini Project -1 (Number Guessing Game)**

**(Language PYTHON)**

* Build a number Guessing Game in which the user selects a range.
* Assume the user selected a range from lower bound of range to Upper bound of range where both lower boundary and upper boundary are integers.
* So a random number in that range is selected by the system where the user needs to guess the number in minimum number of guesses.

**Algorithm for Number Guessing Game:**

1. **Start of the Game:**

• Print a welcome message and game instructions.

• Initialize necessary variables (X, Y, secret\_number, attempts, guessed).

1. **Input Validation Loop:**

• Prompt the user to enter the lower bound (X) and upper bound (Y) of the number range.

• Use a while loop with a try-except block to ensure:

▪ The user inputs are valid integers (int() function).

▪ Y is greater than X to form a valid range.

* If input is invalid (ValueError), prompt user to enter integers again.

**3. Generate Secret Number:**

• Use random.randint(X, Y) to generate a random integer (secret\_number) within the specified range [X, Y].

• Print a message indicating the range of numbers the secret number is within.

**4.Game Loop:**

• Use a while loop with condition not guessed to continue until the player guesses the correct number.

• Inside the loop:

▪ Prompt the user to enter their guess (guess).

▪ Use try-except to handle non-integer inputs and prompt user to enter an integer. ▪ Compare guess with secret\_number:

▪ If guess is less than secret\_number, print "Too low! Try a higher number."

▪ If guess is greater than secret\_number, print "Too high! Try a lower number."

▪ If guess equals secret\_number, set guessed to True, print a congratulatory message, and display the number of attempts (attempts).

**5. Play Again Prompt:**

• After guessing correctly (guessed is True), prompt the user if they want to play again (play\_again).

• Convert play\_again to lowercase for consistency (play\_again.lower()).

• If play\_again is "yes", call number\_guessing\_game() again to start a new game.

• If play\_again is "no" or any other input, print a farewell message and end the game. **6. End of Game:**

• Print "Thank you for playing!" to indicate the end of the game session.

**Pseudo code:**

Function: remove\_match\_char(list1, list2)

1. Find Common Characters:
   * Determine common characters between list1 and list2 using set intersection.
2. If No Common Characters:
   * Concatenate list1, a border marker "\*", and list2.
   * Return [concatenated\_list, False].
3. Remove Common Characters:
   * For each common character:
     + While the character is in list1, remove it from list1.
     + While the character is in list2, remove it from list2.
4. Concatenate Lists:
   * Concatenate list1, a border marker "\*", and list2 into list3.
5. Return Result:
   * Return [list3, True].

Main Program:

1. Input Processing for Player 1:
   * Input name p1.
   * Convert p1 to lowercase.
   * Remove all spaces from p1.
   * Convert p1 to a list of characters, p1\_list.
2. Input Processing for Player 2:
   * Input name p2.
   * Convert p2 to lowercase.
   * Remove all spaces from p2.
   * Convert p2 to a list of characters, p2\_list.
3. Initialize Flag:
   * Set proceed to True.
4. Remove Common Characters Until No Common Characters Remain:
   * While proceed is True:
     + Call remove\_match\_char(p1\_list, p2\_list) and store the result in ret\_list.
     + Extract concatenated list from ret\_list as con\_list.
     + Extract flag from ret\_list as proceed.
     + Find the index of "\*" in con\_list.
     + Split con\_list into p1\_list (characters before "*") and p2\_list (characters after "*").
5. Calculate Remaining Characters:
   * Calculate count as the sum of the lengths of p1\_list and p2\_list.
6. Determine Relationship Status:
   * Initialize result with the list ["Friends", "Love", "Affection", "Marriage", "Enemy", "Siblings"].
   * While the length of result is greater than 1:
     + Calculate split\_index as (count % len(result)) - 1.
     + If split\_index is non-negative:
       - Split result into right (elements after split\_index) and left (elements before split\_index).
       - Concatenate right and left to form the new result.
     + Otherwise:
       - Truncate result to all but the last element.
7. Print Final Relationship Status:
   * Output the relationship status from result[0].

**CODE:**

1. # Function to remove common characters from two lists

2. def remove\_match\_char(list1, list2):

3. common\_chars = set(list1) & set(list2) # Find common characters

4.

5. if not common\_chars:

6. # If no common characters, return lists concatenated with "\*"

7. return [list1 + ["\*"] + list2, False]

8.

9. # Remove common characters from both lists

10. for char in common\_chars:

11. while char in list1:

12. list1.remove(char)

13. while char in list2:

14. list2.remove(char)

15.

16. # Concatenate lists with "\*" as a separator

17. list3 = list1 + ["\*"] + list2

18.

19. # Return concatenated list and True flag

20. return [list3, True]

21.

22. # Main driver code

23. if \_\_name\_\_ == "\_\_main\_\_":

24. # Take first name input and process

25. p1 = input("Player 1 name: ").lower().replace(" ", "")

26. p1\_list = list(p1)

27.

28. # Take second name input and process

29. p2 = input("Player 2 name: ").lower().replace(" ", "")

30. p2\_list = list(p2)

31.

32. proceed = True

33.

34. # Process names until no common characters remain

35. while proceed:

36. ret\_list = remove\_match\_char(p1\_list, p2\_list)

37. con\_list = ret\_list[0]

38. proceed = ret\_list[1]

39.

40. # Find the index of "\*"

41. star\_index = con\_list.index("\*")

42.

43. # Update lists based on position of "\*"

44. p1\_list = con\_list[:star\_index]

45. p2\_list = con\_list[star\_index + 1:]

46.

47. # Count total remaining characters

48. count = len(p1\_list) + len(p2\_list)

49.

50. # FLAMES acronym list

51. result = ["Friends", "Love", "Affection", "Marriage", "Enemy", "Siblings"]

52.

53. # Determine relationship status based on count

54. while len(result) > 1:

55. split\_index = (count % len(result) - 1)

56. if split\_index >= 0:

57. result = result[split\_index + 1:] + result[:split\_index]

58. else:

59. result = result[:len(result) - 1]

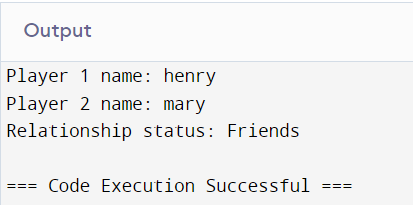
60.

61. # Print final result

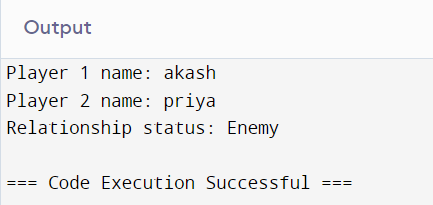
62. print("Relationship status:", result[0])

63.

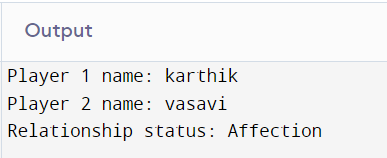
**OUTPUT: 01**

****

**OUTPUT:02**

****

**OUTPUT: 03**

****