Solution Explanation for Padlock Problem

Problem Recap
Farmer John wants to find the best five-letter word to use as the third row on his padlock to maximize the number of valid word combinations. The padlock consists of three rows, each with five letter discs that can rotate to form different combinations.

- **Fixed Rows:** "BELLA" and "LOVES"
- **Flexible Row:** A word of choice to maximize valid words

The goal is to find the third row word that allows the most valid words to be created when each disc is rotated.

Solution Approach and Explanation

Step 1: Define the Problem Constraints

1. Fixed Rows: The first two rows are fixed with specific letters.
2. Flexible Row: The third row can be any five-letter word from the valid words list.
3. Valid Words List: A list of valid five-letter words provided for reference.

Step 2: Generate Combinations

1. **Candidate Word Selection:** Choose a candidate word for the flexible row from the valid words list.
2. **Rotate Each Disc:** For each letter position in the chosen word, rotate the disc to create new letter combinations in that position.

Intentionally left blank

Python Code to generate combinations*(without using* `itertools`*)*

```
def generate_combinations(flexible_row):
    combinations = []
    for i in range(3):  # Rotate first letter position
        for j in range(3):  # Rotate second letter position
            for k in range(3):  # Rotate third letter position
                for l in range(3):  # Rotate fourth letter position
                    for m in range(3):  # Rotate fifth letter position
                        # Create a new combination by modifying each
position
                        new_word = (
                            flexible_row[0][i] +
                            flexible_row[1][j] +
                            flexible_row[2][k] +
                            flexible_row[3][l] +
                            flexible_row[4][m]
                        )
                        combinations.append(new_word)
    return combinations
```

3. **Combination Example:**
    - Suppose “SHEDS” is chosen as the flexible row.
    - Rotate each letter to create combinations like “SHELL,” “SOLES,” etc.

NOTE: The candidate words for the flexible row are selected exclusively from the valid words list. This ensures all generated combinations can be cross-checked against this list for validity.

Step 3: Check Validity of Each Combination

1. **Verify Against Valid Words List:** Each generated word is checked against the list of valid words.
2. **Count Valid Matches:**
    - For each word combination that matches a word in the valid words list, increment the count.

Python Code to check validity:

```
python
Copy code
def count_valid_combinations(flexible_row, valid_words):
    valid_count = 0
    combinations = generate_combinations(flexible_row)
    for word in combinations:
        if word in valid_words:
            valid_count += 1
    return valid_count
```

3. **Example with “SHEDS”:**
    - Valid matches: “BELLS,” “SOLES,” “SEEDS,” etc.
    - *Note:* Only unique valid combinations are counted, and repeated valid words from fixed rows aren’t counted again.

Step 4: Repeat for Each Candidate Word

1. **Try Different Words:** Repeat the above steps for each word in the valid words list as the candidate for the flexible row.
2. **Track Best Word:** Record the number of valid words generated for each candidate and identify the word that results in the highest count of valid combinations.

Python Code to find the optimal word:

```
best_word = ""
max_valid_count = 0

for word in valid_words:
    valid_count = count_valid_combinations(word, valid_words)
    if valid_count > max_valid_count:
        best_word = word
        max_valid_count = valid_count
```

Step 5: Select the Optimal Word

1. **Choose the Word with the Highest Count:** The word that produces the highest number of valid words upon combination generation is selected as the optimal choice for the flexible row.
2. **Final Verification:** Confirm that this word offers a better count than any other in the list.