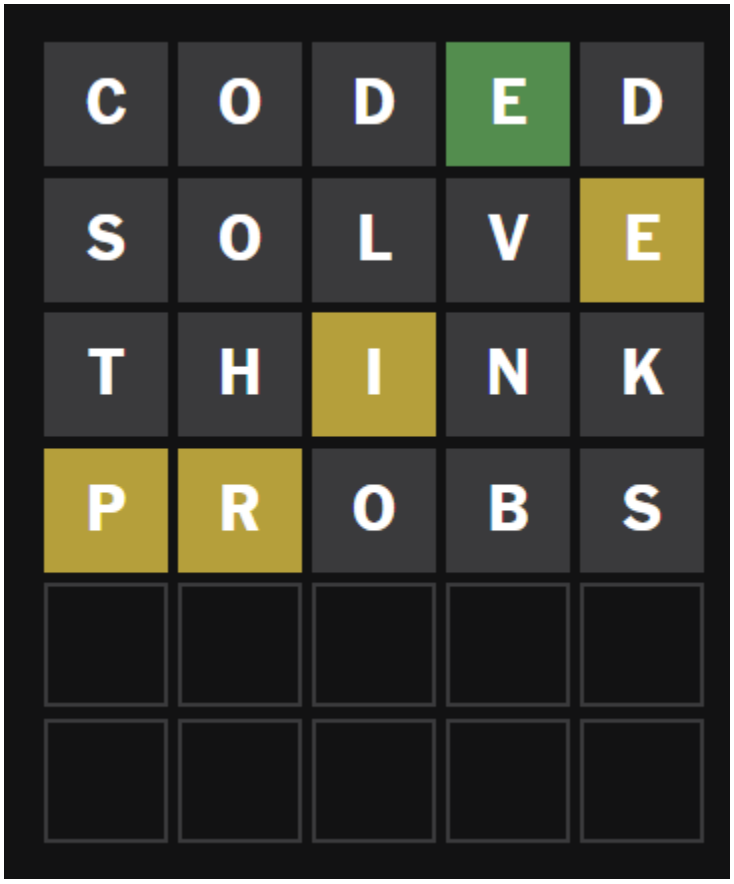
APS106 Design Problem
Week 11: Wordle



Problem Background

Your good friend just introduced you to Wordle, the New York Times word-guessing game that went viral during the pandemic. As a problem-solver, this game has captivated your attention, but alas the NY Times only gives you one word a day, limiting how much you can be playing this game.

You realize that with your recent knowledge of string manipulation you know enough to be dangerous and design your own (and maybe better!) Wordle game in Python. To design this game, you will be relying on your knowledge of strings and string methods, while and for loops, and escape characters for print formatting.

Make sure your code is efficient, well-thought-out, and accounts for all possibilities so can start to Wordle as soon as possible.

Here is a link to the original Wordle game online, play here: https://www.nytimes.com/games/wordle/index.html

If you wanted to read a bit about Wordle instead, plus an infographic: https://www.cnet.com/how-to/wordle-explained-what-you-need-to-know-about-the-viral-word-game/

1. Define the problem

This is a word guessing game relying on string inputs, where we have to have an acceptable (real, 5-letter) word which is the solution, have acceptable user guesses, allow the user up to 6 guesses, and indicate to the user the correctness of their guess by color coding the letters, based on if they are correct and in correct position (green), the correct letter in wrong position (yellow) or not in the word (grey). The final program will obey the game's rules and account for all possible scenarios to be as comprehensive as possible.

The difficult part of this programming exercise will be to define all the checks on the user input and indicating to the end user how close they are to the correct solution. Escape characters will be used to do the color coding part of the game, but you should still consider how to make your program respond once the user gets the correct guess (i.e. if you are using a loop to check the guess each time, how could you indicate when the game should stop?).

2. Define test cases

Firstly, using the dictionaries make sure that user input for solution and any guesses are real words, with 5 letters. Make sure to account for incorrect user inputs, by asking for a new input (and checking that one as well).

Make sure that the user gets 6 attempts to guess the word.

What happens if the user gets the word right? Does the game end or does it continue to use their remaining attempts?

Is your string of incorrect guesses containing the correct letters?

3. Generate solutions

Based on what we have learned so far, our program must:

1. Use input statements to get a solution and user guesses
2. Use loops to keep track of the # of attempts
3. Use loops to check the guess string compared to the solution string and display this result
4. Have a string keeping track of all the incorrect letters (i.e. guessed letters not in solution).
5. Finish the game if either the user has no more attempts, or the correct word is guessed

An algorithmic plan would be:

1. Get a random solution from a list of valid words and a word selection from the user
2. Compare each letter of guess to the solution letters.
3. Indicate which letters are correct, which are incorrect in a clear print output to users.
4. Keep an internal score, to indicate if all 5 letters in a guess are correct.
5. Ensure attempts are tracked, as well as all incorrect letters in guesses (see final image at bottom of notebook).

What about a Programming Plan? How can we implement this efficiently and cleanly?

1. Get input from the users and store in two variables.
2. Compare inputs and compute the results. Track # of guesses and determine way to indicate 100% match between guess and solution.
3. Display output!

4. Select a solution

Does the best solution you developed as an algorithm plan make sense?

5. Implement and test your solution

Program your solution step-by-step. Are there things you need to add to or change in your algorithm plan?