We chose to code the connect four game as we had a really good idea for how to implement the turtle graphics. In addition, we wanted to tackle something which we could implement classes in as we haven't learned about classes during lectures. We designed our program with a large Connect4Board class with separate methods to handle certain aspects of the game such as switching turns, placing pieces etc.

GAME HIERARCHY

Global var COLORS: list[str]

- ['white', 'red', 'yellow']
- We can index this list to convert piece ids to colors since it is easier to work with numbers such as 0, 1, and 2 in our program rather than using strings every step of the way when we really only need it for turtle drawing calls
  - 0 corresponds to no piece, thus WHITE
  - 1 corresponds to RED piece
  - 2 corresponds to YELLOW piece

- Class Connect4Board
  - Instance var matrix:
    - 2D list of integers to store board state
      - 0 = no piece
      - 1 = has red piece
      - 2 = has yellow piece
    - Each row in the matrix corresponds to a column in the board
      - Makes it easier to perform checks and place pieces, since pieces gravitate to the lowest possible position in a connect4 column
  - Instance var turn
    - Either 1 or 2 meaning either red or yellow respectively
  - constructor __init__(NUM_ROWS, NUM_COLS)
    - Generate a NUM_COLS x NUM_ROWS list to store the state of our board after every move and store it in instance variable matrix
    - Initialize turn to 0
  - instance method place_piece (column index)
    - Try placing a piece at the given column index and return boolean value whether we were able to successfully place it there or not
    - For loop to check every position in that column starting from the bottom
      - If there is already a piece, continue checking upwards
      - If there is no piece, place the current player's piece there and return True since we were able to successfully place a piece
        - Use the fill_piece function to update the turtle window
        - We can access the current player's piece by using the instance variable turn and place that in the matrix position
    - If we made it out of the loop without returning True, return False
  - instance method check_dir (x, y, dx (delta x), dy (delta y))
    - Recursively check through a specified direction by starting at the point x, y in the matrix and incrementing those indices by delta x and delta y after

each check. This will traverse through all the pieces in that direction until we meet an exit condition, which is either a win or our indices being out of bounds of the list.
    - Store the last piece we found (can be initialized to either 1 for red or 2 for yellow, doesn't matter at the start) and the current streak (intiialized to 0) in two local integer variables
    - While loop until our streak is at least 4 (4 in a row) or our indices are now out of bounds
        - If the piece in the current position is the same as the last piece we found, we increment our streak by 1
        - Otherwise, we set our last piece variable to the current piece and reset our streak to 1, since this current piece counts as the first in that streak
        - Then, regardless of whether we found the same piece or not, we increment our x and y values by delta x and delta y respectively
    - Return
        - The value of the last piece variable if the streak is at least 4, meaning we met a win condition
        - 0 if the streak is less than 4, meaning no win condition was met in the given direction
- instance method <u>check_win</u>
    - Use the check_dir function to check every vertical, horizontal, and diagonal section of the board for a win condition
    - Initialize a local integer variable to store the winner of the game (either 0, 1, or 2 for no one, red, or yellow respectively)
    - Verticals and horizontals are easy
        - Verticals: check_dir(0, y, dx=1, dy=0) for 0 <= y < NUM_ROWS
        - Horizontals: check_dir(x, 0, dx=1, dy=1) for 0 <= x <= NUM_COLS
    - Diagonals are on the other hand a bit trickier
        - If we map it out
            - We see that in each diagonal direction, we don't need to check the first 3, since none of them will contain enough pieces to result in a connect 4
            - We also see that there are 4 different types of diagonals that we can iterate through, so we'll have to write all of those out
        - check_dir(x, 0, dx=-1, dy=1) for 3 <= x < NUM_COLS
        - check_dir(x, 0, dx=1, dy=1) for 0 <= x < NUM_COLS - 3
        - check_dir(0, y, dx=1, dy=-1) for 3 <= y < NUM_ROWS
        - check_dir(0, y, dx=1, dy=1) for 0 <= y < NUM_ROWS - 3
    - For each of these check_dir calls, we update the winner function with the returned result if it is 0, otherwise we retain its value. We do this since if we already found a winner, we don't want to reset the winner variable to 0 just because another section doesn't contain a win condition

        - Lastly, we check if every position in the board is already filled with a piece, in which case we return 3 for a draw
        - Otherwise, we return the value of the winner variable (0,1, or 2)
    - instance method save
        - Save the current board configuration and the current turn in a text file
    - instance method load
        - Check if a save file exists
            - If so, load the current board configuration and current turn from that text file and update instance variables accordingly
    - instance method game_to_world (x_game, y_game)
        - This function will be useful for drawing to the screen using turtle graphics, since turtle works with a different coordinate space than our connect4 board
        - Given the x and y coordinates that specify a position in our matrix, we return a tuple (x_world, y_world) with those x and y coordinates in their corresponding positions in turtle space
    - instance method fill_piece(x_game, y_game, piece)
        - Use turtle graphics to draw a circle on the screen in the position corresponding to the specified board position and fill the circle with the color corresponding to the given piece
        - Use game_to_world to get the position in the screen
            - Position the turtle in that position
            - Set the fill color to the appropriate color by indexing our COLORS list by the integer piece id
            - Use the circle function to draw a circle in the position
    - instance method draw
        - Draws the entire board to the screen
        - Clear the screen and set the background color to blue (typical connect 4 boards are blue)
        - Iterate through the board matrix and call the fill_piece function for every element in the matrix