Lab 10 Feedback

- Adhere to interface
  - Accepted parameter types
  - Type of what is returned
- Use methods you've already written
  - Example: use `addPerson` in `addPeople`
- What to return when errors

Lab 11: Three Parts

- Linux practice:
  - Using the **WC** command
- Social Network extensions
  - Binary search – find people with a certain name
  - UI: add search functionality
- Two-dimensional lists
  - Including Connect Four

WC Command

- **WC**: Word Count
  - Count up the lines of Social Network code from Lab 10
  - Compare with code for this assignment
- Specific directions are in the lab

Social Network, Extended

- Searching Overview
  - Allows you to search for people by their name– lowercased for more intuitive results
  - Update `SocialNetwork` class and UI appropriately

Summary of Modifications to Binary Search

- Add a search method
  - Takes as parameter the network to search for
- Check the *name* of the Person that is at the midpoint, lowercased
- After found, add to the list of Persons who match
  - Get the Persons before and after that person in the list that have the same name and add to list
- Represent (in method) and handle (in UI) when no people with that name
- For "most intuitive" results:
  - Lowercase the key
    - Changes algorithm again slightly

Search

- What does your implemented binary search algorithm become in "the worst case"?
- Assumption: only a few people will have the same name
  - Otherwise, may fbe just as well to use linear search

SocialNetwork Code

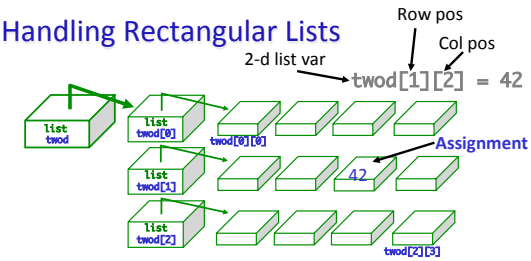
- Fix the major problems in your code first
- Or, use the code in the `handouts/lab11/solution` directory
  - person.py, social.py, facespace.py

2D LISTS

Review

- How do you create a 2D list?
- How do you get the 2nd element in the 3rd "row" of a list?
- How do you find the number of lists in a 2D list?
- How do you find the number of elements in one of those lists?

Handling Rectangular Lists



- What does each component of `twod[1][2]` mean?
- How many rows does `twod` have, in general?
  - `rows = len(twod)`
- How many columns does `twod` have, in general?
  - `cols = len(twod[0])`

Generalize Creating a 2D List

- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in list to 0

```
def create2DList(rows, cols):
   twodlist = [ ]
   # for each row
   for row in range( rows ):
      row = [ ]
      # for each column, in each row
      for col in range( cols ):
          row.append(0)
      twodlist.append(row)
   return twodlist
```


Game Board for Connect Four

- 6 rows, 7 columns board
- Players alternate dropping red/black checker into slot/column
- Player wins when have four checkers in a row vertically, horizontally, or diagonally

How do we represent the board as a 2D list, using a graphical representation?

Game Board for Connect Four

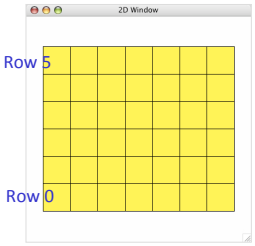
- How to represent board in 2D list, using graphical representation?

Number	Meaning	Color
0	Free	Yellow
1	Player 1	Red
2	Player 2	Black

Game Board for Connect Four

- How to represent board in 2D list, using graphical representation?

Number	Meaning	Color
0	Free	Yellow
1	Player 1	Red
2	Player 2	Black



Connect Four (C4): Making moves

- User clicks on a column
  - "Checker" is filled in at that column

```
# gets the column of where user clicked
col = csplot.sqinput()
```


Problem: C4 - Valid move?

- Need to enforce valid moves
  - In physical game, run out of spaces for checkers if not a valid move
- How can we determine if a move is valid?
  - How do we know when a move is ***not*** valid?

ConnectFour Class

- Play the game method implementation
  - Repeat:
    - Get input/move
    - Check if valid mo
    - Make move
    - Display board
    - Check if win
    - Change player

```
won = False
player = ConnectFour.PLAYER1

while not won:
    print("Player %d's move" % player)
    if player == ConnectFour.PLAYER1:
        col = self._userMakeMove()
    else: # computer is player 2
        # pause because otherwise move happens too
        # quickly and looks like an error
        sleep(.75)
        col = self._computerMakeMove()

    row = self.makeMove(player, col)
    self.showBoard()
    won = self._isWon(row, col)

    # alternate players
    player = player % 2 + 1
```


Connect Four (C4): Making moves

- User clicks on a column
  - "Checker" is filled in at that column

```
# gets the column of where user clicked
col = csplot.sqinput()
```

```
def _userMakeMove(self):
    """ Allow the user to pick a column."""
    col = csplot.sqinput()
    validMove = self._isValidMove(col)
    while not validMove:
        print("NOT A VALID MOVE.")
        print("PLEASE SELECT AGAIN.")
        print()
        col = csplot.sqinput()
        validMove = self._isValidMove(col)
    return col
```

Problem: C4 - Making a Move

- The player clicks on a column, meaning that's where the player wants to put a checker
- How do we update the board?

Looking Ahead

- Bring your final exam envelopes to me by Friday
- Bring your final exam questions Friday

Thanks to Azmain, Perry, Sarah Anne, and Shane for their help this semester!