Objectives

- Two-dimensional lists

Review

- What are the two types of search we discussed?
  - How do they work?
- What are the tradeoffs between using linear search and binary search?
- What was our (final) sorting algorithm?

Algorithm: Merge Sort

```
def mergeSort( listOfNumbers ):
    if len(listOfNumbers) == 2: # base case
        # sort those two numbers
        if listOfNumbers[0] > listOfNumbers[1]:
            temp = listOfNumbers[0]
            listOfNumbers[0] = listOfNumbers[1]
            listOfNumbers[1] = temp
        return listOfNumbers
    firsthalf = listOfNumbers[:len(listOfNumbers)//2 ]
    secondhalf = listOfNumbers[len(listOfNumbers)//2:]
    sortedFirst = mergeSort( firsthalf )
    sortedSecond = mergeSort( secondhalf )
    whole = merge( sortedFirst, sortedSecond )
    return whole
```


2D LISTS

Lists

- We've used lists that contain
  - Integers
  - Strings
  - Cards (Deck class)
  - Persons (your Person class)
- We discussed that lists can contain multiple types of objects within the same list
  - Wheel of Fortune: ["Bankrupt", 250, 350, …]
- Lists can contain *any* **type** of object
  - Even **LISTS**!

Review of Regular (1D) Lists

- Create a list

```
onedlist = [ 7, -1, 23 ]
```

Elements in the list

- How do we find the number of elements in the list?
- How can we find the value of the third element in the list?

Review of Regular (1D) Lists

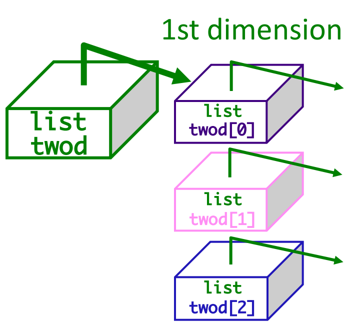
- Create a list `onedlist = [ 7, -1, 23 ]`

Elements in the list

- `len(onedlist)` is 3
- `onedlist[2]` is 23

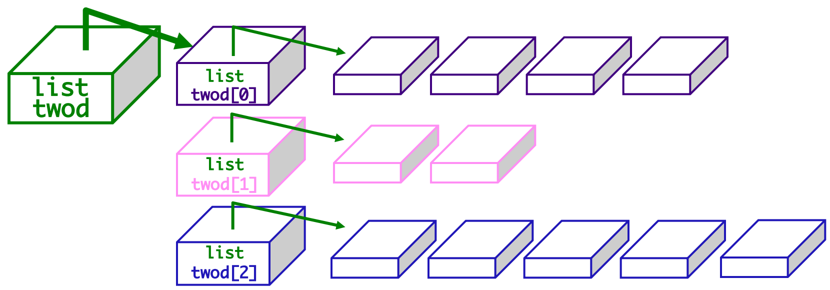
A List of Lists: 2-dimensional List

```
             twod[0]   twod[1]    twod[2]
twod = [ [1,2,3,4], [5,6], [7,8,9,10,11] ]
```



A List of Lists: 2-dimensional lists

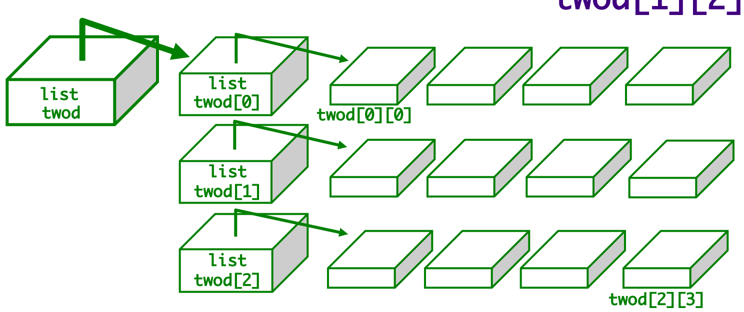
```
twod = [ [1,2,3,4], [5,6], [7,8,9,10,11] ]
```



- "Rows" within 2-dimensional list do **not** need to be the same length
- However, it's often easier if they're the same length!
  - We'll focus on "**rectangular**" 2-d lists

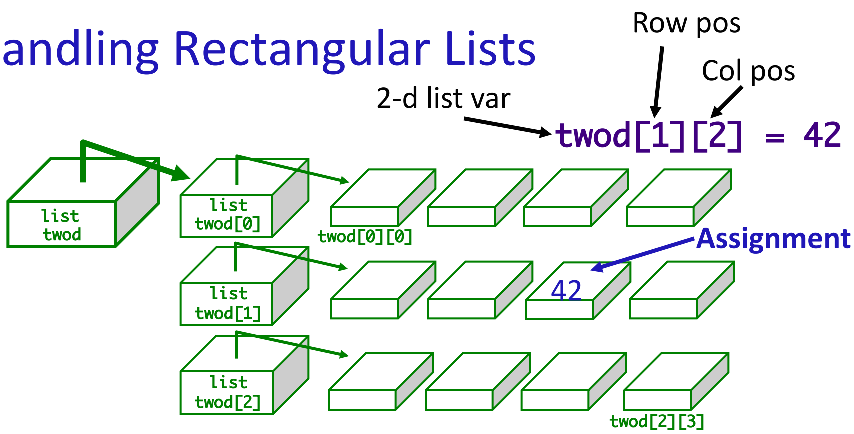
Handling Rectangular Lists

```
twod[1][2] = 42
```



- What does each component of `twod[1][2]` mean?
- How many rows does `twod` have, in general?
- How many columns does `twod` have, in general?

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])`

Practice

Starting with the 2d list **twod** shown here, what are the values in **twod** after running this code?

twod Before

	col 0	col 1	col 2	col 3
row 0 →	1	2	3	4
row 1 →	5	6	7	8
row 2 →	9	10	11	12

```
def mystery(twod):
    """ 'run' this on twod, at right """
    for row in range( len(twod) ):
        for col in range( len(twod[0]) ):
            if row == col:
                twod[row][col] = 42
            else:
                twod[row][col] += 1
```

twod After

Practice

Starting with the 2d list **twod** shown here, what are the values in **twod** after running this code?

twod Before

	col 0	col 1	col 2	col 3
row 0	1	2	3	4
row 1	5	6	7	8
row 2	9	10	11	12

```
def mystery(twod):
    """ 'run' this on twod, at right """
    for row in range( len(twod) ):
        for col in range( len(twod[0]) ):
            if row == col:
                twod[row][col] = 42
            else:
                twod[row][col] += 1
```

twod After

42	3	4	5
6	42	8	9
10	11	42	13

Typical Use of 2D List

1. Initialize the 2D list
    1. Make all the "spots" available in the list
    2. Initialize those spots to some value
2. Fill in the spots as appropriate.

Example: Creating a 2d List

```
twod = [ ]
```

- Create a row of the list

  `row = [1, 2, 3, 4]` or `row = list(range(1,5))`

- Then append that row to the list

```
twod.append( row )
print(twod)
```

  - [[1, 2, 3, 4]]

- Repeat

```
row = [1, 2, 3, 4]
twod.append( row )
print(twod)
```

  - [[1, 2, 3, 4], [1, 2, 3, 4]]

Generalize Creating a 2D List

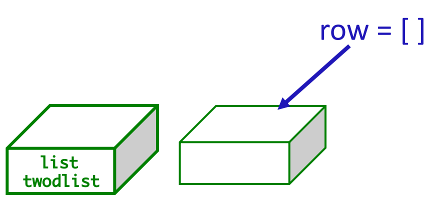
- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in (sub) list to 0

Generalize Creating a 2D List

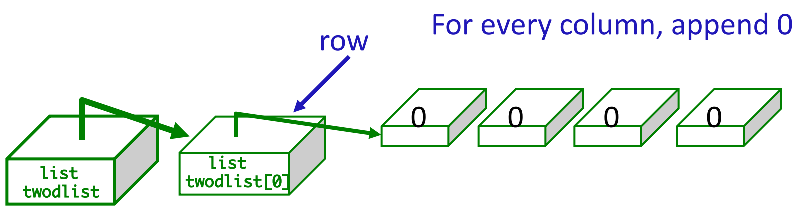
- Create a function that returns a 2D list with width ***cols*** and height ***rows***
  - Initialize each element in list to 0

```
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
```


How Does This Work?

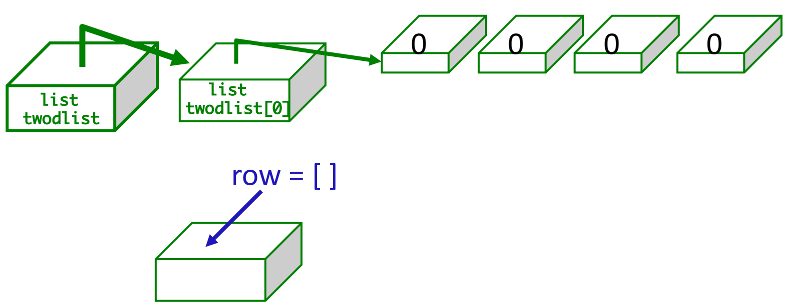


How Does This Work?

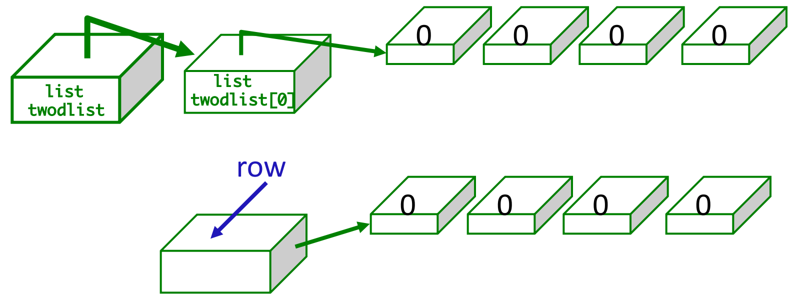


Append row to twodlist

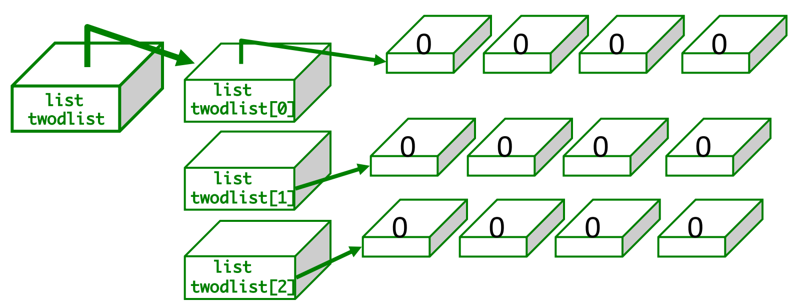
How Does This Work?



How Does This Work?



How Does This Work?



Incorrect: Creating a 2D List

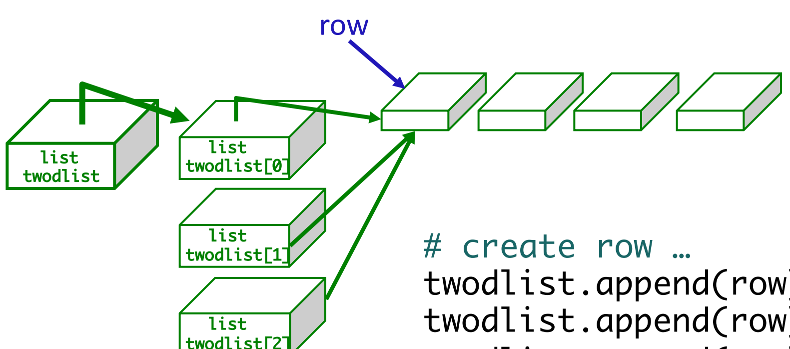
- The following code **won't** work. Why?
- Explain output from example program

```
def noCreate2DList(rows, cols):
    twodlist = [ ]
    row = [ ]
    # create a row with appropriate columns
    for col in range( cols ):
        row.append(0)
    # append the row rows times
    for r in range( rows ):
        twodlist.append(row)
    return twodlist
```

twod_exercises.py

All Rows Pointing at Same Block of Memory

- Each row points to the **same** row in memory

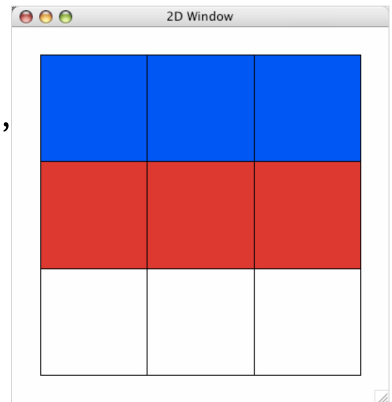



```
# create row …
twodlist.append(row)
twodlist.append(row)
twodlist.append(row)
```

Graphical Representation of 2D Lists

- Module: `csplot`
- Allows you to visualize your 2D list
  - Numbers are represented by different colors

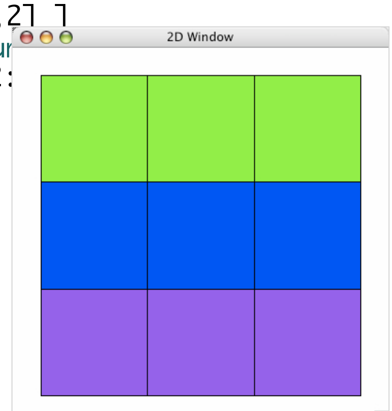
```
import csplot
…
# create 2D list…
twodlist=[ [0,0,0], [1,1,1], [2,2,
# display list graphically
csplot.show(twodlist)
```



Graphical Representation of 2D Lists

- Can assign colors to numbers

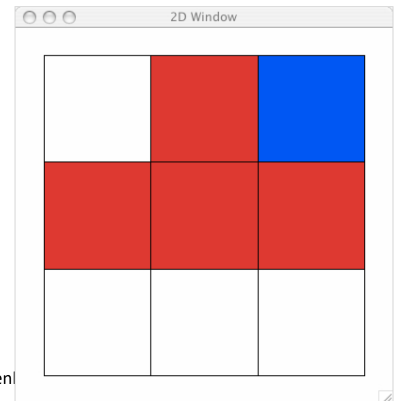
```
import csplot
…
# create 2D list…
twodlist=[ [0,0,0], [1,1,1], [2,2,2] ]
# create optional dictionary of nur
numToColor={0:"purple", 1:"blue", 2:
csplot.show(twodlist, numToColor)
```



Graphical Representation of 2D Lists

```
matrix = [[0,0,0], [1,1,1], [0,1,2]]
```

What values map to which colors by default?



Graphical Representation of 2D Lists

- Note that representation of rows is backwards from how we've been visualizing

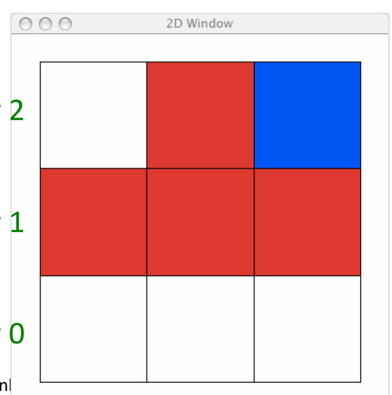
```
matrix = [[0,0,0], [1,1,1], [0,1,2]]
```

What values map to which colors by default?

Row 2

Row 1

Row 0



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

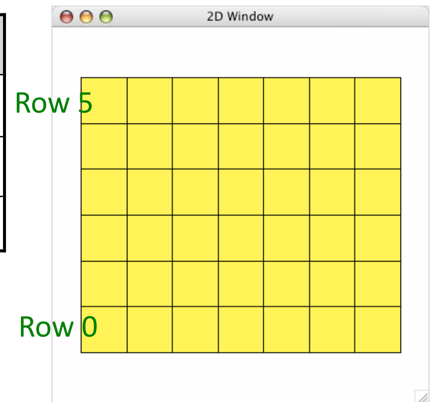
- How to represent board in 2D list, using graphical representation?

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



ConnectFour Class

- What is the data associated with the class?
- What methods should we implement?

ConnectFour Class

- Data
  - Board + constants
    - 6 rows, 7 columns, all FREE to start
- Methods
  - Constructor
  - Display the board
  - Play the game
  - Get input/move from user
  - Check if valid move
  - Make move
  - Check if win

ConnectFour Constants

```
class ConnectFour:
    """ Class representing the game Connect Four. """

    # Represent different values on the board
    FREE = 0
    PLAYER1 = 1
    PLAYER2 = 2

    # Represent the dimensions of the board
    ROWS = 6
    COLS = 7
```

ConnectFour Class

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

```
def play(self):
    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 - 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?

Problem: C4 - Valid move?

- Solution: check the “top” spot
  - If the spot is FREE, then it’s a valid move

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

- Lab 11 – Tomorrow
- Broader Issue: Facebook – Friday
- Bring Exam envelopes