

Objectives

- Two-dimensional lists

April 4, 2016

Sprengle - CSCI111

1

Review

- What are the tradeoffs between using linear search and binary search?

April 4, 2016

Sprengle - CSCI111

2

2D LISTS

April 4, 2016

Sprengle - CSCI111

3

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!**

April 4, 2016

Sprengle - CSCI111

4

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?

April 4, 2016

Sprengle - CSCI111

5

Review of Regular (1D) Lists

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

Elements in the list

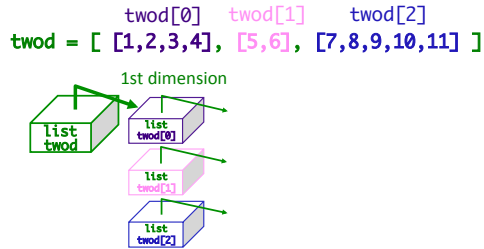
- `len(onedlist)` is 3
- `onedlist[2]` is 23

April 4, 2016

Sprengle - CSCI111

6

A List of Lists: 2-dimensional List



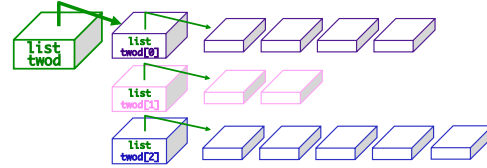
April 4, 2016

Sprengle - CSCI111

7

A List of Lists: 2-dimensional lists

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



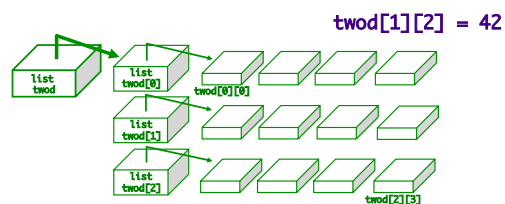
- “Rows” within 2-dimensional list do **not** need to be the same length
- However, it’s often easier if they’re the same length!
 - We’ll focus on “rectangular” 2-d lists

April 4, 2016

Sprengle - CSCI111

8

Handling Rectangular Lists



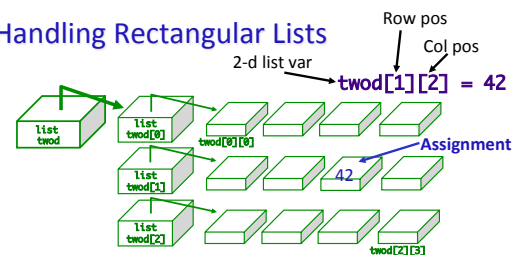
- What does each component of $twod[1][2]$ mean?
- How many rows does $twod$ have, in general?
- How many columns does $twod$ have, in general?

April 4, 2016

Sprengle - CSCI111

9

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?
 - $cols = len(twod[0])$

April 4, 2016

Sprengle - CSCI111

10

Practice

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

```
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 Before

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

row 0 →
row 1 →
row 2 →

col 0 col 1 col 2 col 3

twod After

April 4, 2016

Sprengle - CSCI111

mystery.py

11

Practice

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

```
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 Before

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

row 0 →
row 1 →
row 2 →

col 0 col 1 col 2 col 3

twod After

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

April 4, 2016

Sprengle - CSCI111

mystery.py

12

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.

April 4, 2016

Sprengle - CSCI111

13

Example: Creating a 2d List

- ```
twod = []
```
- Create a row of the list  
`row = [1, 2, 3, 4]`
  - 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]]`

April 4, 2016

Sprengle - CSCI111

14

## 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

April 4, 2016

Sprengle - CSCI111

15

## 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

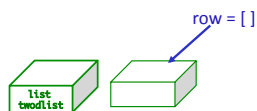
```
def create2DList(rows, cols):
 twodlist = []
 # for each row
 for row in range(rows):
 row = []
 # for each column, in each row
 for col in range(cols):
 row.append(0)
 twodlist.append(row)
 return twodlist
```

April 4, 2016

Sprengle - CSCI111

16

## How Does This Work?

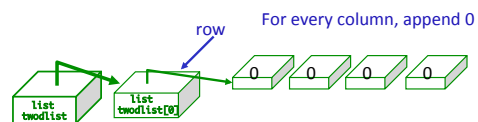


April 4, 2016

Sprengle - CSCI111

17

## How Does This Work?



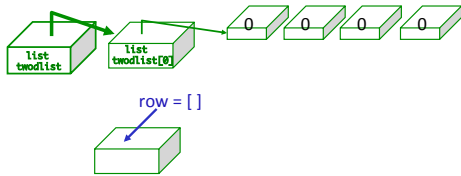
Append row to twodlist

April 4, 2016

Sprengle - CSCI111

18

## How Does This Work?

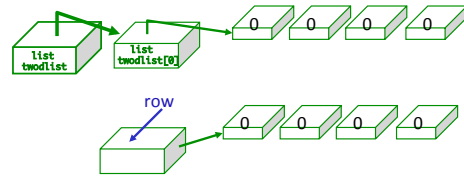


April 4, 2016

Sprengle - CSCI111

19

## How Does This Work?

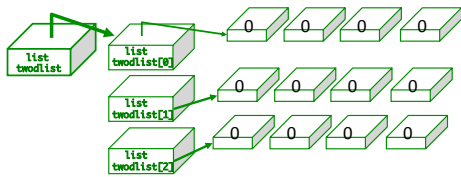


April 4, 2016

Sprengle - CSCI111

20

## How Does This Work?



April 4, 2016

Sprengle - CSCI111

21

## 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

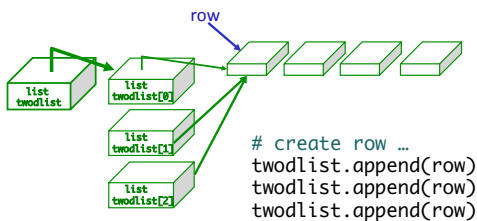
April 4, 2016

Sprengle - CSCI111

22

## 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)
```

April 4, 2016

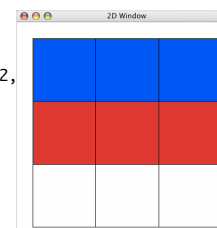
Sprengle - CSCI111

23

## Graphical Representation of 2D Lists

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

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



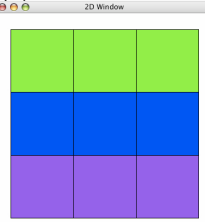
April 4, 2016

Sprengle - CSCI111

## Graphical Representation of 2D Lists

- Can assign colors to numbers

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

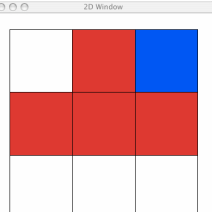


April 4, 2016 Sprengle - CSC1111

## Graphical Representation of 2D Lists

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

What values map to which colors?



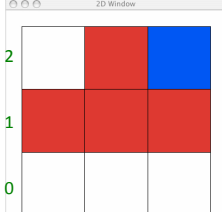
April 4, 2016 Sprengle 26

## 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?



April 4, 2016 Sprengle 27

## 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?

April 4, 2016 Sprengle - CSC1111 28

## Game Board for Connect Four

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

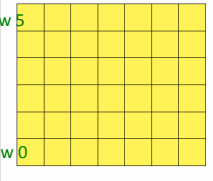
| Number | Meaning  | Color  |
|--------|----------|--------|
| 0      | Free     | Yellow |
| 1      | Player 1 | Red    |
| 2      | Player 2 | Black  |

April 4, 2016 Sprengle - CSC1111 29

## 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  |



Row 5  
Row 0

April 4, 2016 Sprengle - CSC1111 30

## ConnectFour Class

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

April 4, 2016

Sprengle - CSC1111

31

## 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

April 4, 2016

Sprengle - CSC1111

32

## 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
```

April 4, 2016

Sprengle - CSC1111

33

## 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
```

April 4, 2016

Sprengle - CSC1111

34

## 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?

April 4, 2016

Sprengle - CSC1111

35

## Problem: C4 - Valid move?

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

April 4, 2016

Sprengle - CSC1111

36

### 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?

April 4, 2016

Sprengle - CSCI111

37

### Looking Ahead

- Lab 11 – Tomorrow
- No Broader Issue for this week

April 4, 2016

Sprengle - CSCI111

38