

Nested Lists

- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

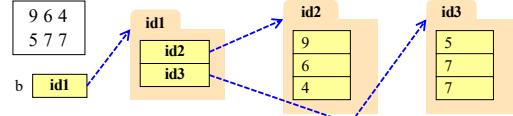
$a = [2, 1]$
 $b = [3, 1]$
 $c = [1, 4, b]$
 $x = [1, a, 5]$

$x = [1, [2, 1], [1, 4, [3, 1]], 5]$

1

How Multidimensional Lists are Stored

- $b = [[9, 6, 4], [5, 7, 7]]$



- b holds name of a one-dimensional list

- Has $\text{len}(b)$ elements

- Its elements are (the names of) 1D lists

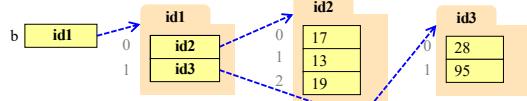
- $b[i]$ holds the name of a one-dimensional list (of ints)

- Has $\text{len}(b[i])$ elements

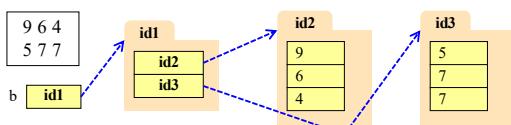
2

Ragged Lists vs Tables

- Ragged is 2d uneven list: $b = [[17, 13, 19], [28, 95]]$



- Table is 2d uniform list: $b = [[9, 6, 4], [5, 7, 7]]$



3

Representing Tables as Lists

Spreadsheet

	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9
4	6	7	8	0

Each row, col has a value

- Represent as 2d list
 - Each table row a list
 - List of all rows
 - Row major order**
- Column major exists
 - Less common to see
 - Limited to some scientific applications

$d = [[8, 4, 7, 3], [4, 8, 9, 7], [5, 1, 2, 3], [4, 1, 2, 9], [6, 7, 8, 0]]$

4

Overview of Two-Dimensional Lists

- Access value at row 3, col 2:

$d[3][2]$

	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9
4	6	7	8	0

- Assign value at row 3, col 2:

$d[3][2] = 8$

- An odd symmetry**

- Number of rows of d : $\text{len}(d)$

- Number of cols in row r of d : $\text{len}(d[r])$

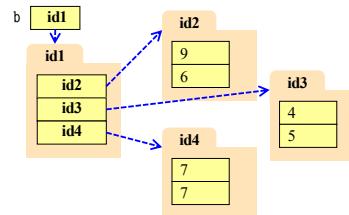
5

Slices and Multidimensional Lists

- Only “top-level” list is copied.

- Contents of the list are not altered

- $b = [[9, 6], [4, 5], [7, 7]]$



6

Shallow vs. Deep Copy

- **Shallow copy:** Copy top-level list
 - Happens when slice a multidimensional list
- **Deep copy:** Copy top and all nested lists
 - Requires a special function: `copy.deepcopy`
- **Example:**

```
>>> import copy
>>> a = [[1,2],[2,3]]
>>> b = a[:]           # Shallow copy
>>> c = copy.deepcopy(a) # Deep copy
```

7

Functions over Nested Lists

- Functions on nested lists similar to lists
 - Go over (nested) list with *for-loop*
 - Use *accumulator* to gather the results
- But two important differences
 - Need **multiple for-loops**
 - One for each part/dimension of loop
 - In some cases need **multiple accumulators**
 - Latter true when result is new table

8

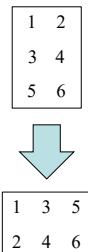
Functions on Nested Lists

```
def all_nums(table):
    """Returns True if table contains only numbers
    Precondition: table is a (non-ragged) 2d List"""
    result = True
    # Walk through table
    for row in table:
        # Walk through the row
        for item in row:
            if not type(item) in [int,float]:
                result = False
    return result
```

9

Transpose: A Trickier Example

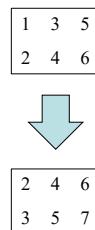
```
def transpose(table):
    """Returns: copy of table with rows and columns swapped
    Precondition: table is a (non-ragged) 2d List"""
    numrows = len(table) # Need number of rows
    numcols = len(table[0]) # All rows have same no. cols
    result = []
    for m in range(numcols):
        row = []
        for n in range(numrows):
            row.append(table[n][m]) # Create a new row list
        result.append(row) # Add result to table
    return result
```



10

A Mutable Example

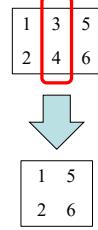
```
def add_ones(table):
    """Adds one to every number in the table
    Preconditions: table is a 2d List,
    all table elements are int"""
    # Walk through table
    for rpos in range(len(table)):
        # Walk through each column
        for cpos in range(len(table[rpos])):
            table[rpos][cpos] = table[rpos][cpos]+1
    # No return statement
```



11

Another Example

```
def strip(table,col):
    """Removes column col from the given table
    Preconditions: table is a (non-ragged) 2d List,
    col valid column"""
    # Walk through table
    for rpos in range(len(table)):
        # Modify each row to slice out column
        table[rpos] = table[rpos][:col] + table[rpos][col+1:]
    # No return statement
```



12