YEAH: Assignment 4

Tori and Kara
Part 1: Lists
**Goal**: Given a list, return a new list of all the elements of the original list that were greater than 10

- If there are no elements greater than 10 or no elements at all, return an empty list
- Create a new list and don't modify the old list
Removing Duplicates from User Input

Goal: Prompt a user for integers until 0 is entered and return a list of the unique integers given

- read_list()
- remove_duplicates(num_list)

Enter value (0 to stop): 2
Enter value (0 to stop): 2
Enter value (0 to stop): 4
Enter value (0 to stop): 2
Enter value (0 to stop): 0

[2, 4]

Hint: Check out this week's section problems if you're stuck
Hint: Think about types
Ziplists
---

Goal: given two lists, pair up numbers (e.g. first element from list1 with first element of list2) and return a list of lists where each sublist is the pairing

- **Step 1:** Make one pairing into a list
- **Step 2:** Add that list to a list of lists

If there are no elements in list1 and list2, just return an empty list

*Hint*: Think of how the two sides of a zipper come together
Ziplist Example

\[
\begin{align*}
\text{lst1} &= [1] \\
\text{lst2} &= [9]
\end{align*}
\]

\[
\begin{align*}
\text{lst1} &= [1, 2] \\
\text{lst2} &= [9, 7]
\end{align*}
\]

\[
\begin{align*}
\text{lst1} &= [1, 9] \\
\text{lst2} &= [2, 7]
\end{align*}
\]
Part 2: Sand
[demo of what Sand should look like]
The Sand World

---

**World**

```
<table>
<thead>
<tr>
<th>'r'</th>
<th>'s'</th>
<th>'r'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>'r'</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>'r'</th>
<th>'r'</th>
<th>'r'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>'s'</td>
<td>'r'</td>
</tr>
</tbody>
</table>
```

```
[[['r', 's', 'r'], ['r', None, 'r']],[['r', None, 'r'], ['r', 's', 'r']]]
```

**Elements**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sand</td>
<td>'s'</td>
</tr>
<tr>
<td>Rock</td>
<td>'r'</td>
</tr>
<tr>
<td>Empty</td>
<td>None</td>
</tr>
</tbody>
</table>
REMEMBER

A grid's x,y coordinates are reversed

Element at x=1, y=0 is at grid[0][1]

For this presentation, when we use (x,y) we are referring to traditional coordinates, not grid coordinates.
Milestone 1: Moving elements in grid

---

do_move(grid, x1, y1, x2, y2)

Take what is at (x1, y1) of the grid, move it to (x2, y2), and set (x1, y1) to be None.

Grid before:

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>'s'</td>
</tr>
<tr>
<td>0   1</td>
</tr>
</tbody>
</table>
```

Grid after:

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>'s'</td>
</tr>
<tr>
<td>0   1</td>
</tr>
</tbody>
</table>
```
Milestone 2: Checking legal moves

There are 3 legal moves:

- moving straight down, if \((x_2, y_2)\) has nothing in it
- moving diagonally left, if the spot immediately to left of \((x_1, y_1)\) is empty and if \((x_2, y_2)\) has nothing in it
- moving diagonally right, if the spot immediately to right of \((x_1, y_1)\) is empty and if \((x_2, y_2)\) has nothing in it

For all of these moves, the destination \((x_2, y_2)\) must be within the grid boundaries
Diagonal Moves

What are the coordinates of destinations for diagonal moves?

\[
(1, 0) \\
(x_1, y_1)
\]
Diagonal Moves

What are the coordinates of destinations for diagonal moves?

\[(1,0)\]  
\[(x_1, y_1)\]

\[(0,1)\]  
\[(x_1 - 1, y_1 + 1)\]

\[(2,1)\]  
\[(x_1 + 1, y_1 + 1)\]
Milestone 3: Gravity

- Step 1: check if straight down is valid
- Step 2: if straight down didn't work, check if you can go diagonal
- Step 3: if no legal moves, don't do anything

Before

\[
\begin{array}{ccc}
'r' & 's' & 'r' \\
'r' & 'r' & 'r'
\end{array}
\]

After

\[
\begin{array}{ccc}
'r' & 's' & 'r' \\
'r' & 'r' & 'r'
\end{array}
\]

- Sand at (1, 0) moved directly down to (1,1)

Before

\[
\begin{array}{ccc}
's' & 'r' \\
'r' & 'r'
\end{array}
\]

After

\[
\begin{array}{ccc}
's' & 'r' & 'r'
\end{array}
\]

- Sand at (1, 0) moved diagonally left to (0,1)
Milestone 4: Loop through the whole grid

- For each \((x,y)\) location, call gravity
- But what order should we iterate over all these locations?
  - ORDER MATTERS

**Iterating from top to bottom**

<table>
<thead>
<tr>
<th>Call gravity on ((0,0))</th>
<th>Nothing changed!</th>
<th>Call gravity on ((0,1))</th>
<th>Sand moved down one</th>
</tr>
</thead>
<tbody>
<tr>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
</tr>
<tr>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Iterating from bottom to top**

<table>
<thead>
<tr>
<th>Call gravity on ((1,0))</th>
<th>Sand moved down one</th>
<th>Call gravity on ((0,0))</th>
<th>Sand moved down one</th>
</tr>
</thead>
<tbody>
<tr>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
</tr>
<tr>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
<td>'s'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

'{}' Changed!
Milestone 4: Loop through the whole grid

- For each \((x,y)\) location, call gravity
- But what order should we iterate over all these locations?
  - ORDER MATTERS

Iterating from top to bottom:
- call gravity on \((0,0)\)
  - Nothing changed!
- call gravity on \((0,1)\)
  - sand moved down one
- call gravity on \((1,0)\)
  - sand moved down one

Iterating from bottom to top:
- call gravity on \((0,0)\)
  - sand moved down one
- call gravity on \((0,1)\)
  - sand moved down one
- call gravity on \((1,0)\)
  - sand moved down one
Milestone 5: Create Brownian motion

---

Giving sand Brownian motion:

```python
num = random.randrange(100)
coin = random.randrange(2)
```

num < brownian
Make sure your functions work in harmony!

- All of your functions should work in harmony with one another
- If you already validate values (error checking) in one function, don't have to do it again in some helper function
Good luck!