More Lists

Chris Gregg
Based on slides by Chris Piech and Mehran Sahami
CS106A, Stanford University
Housekeeping

• Assignment #3 due **tomorrow**
• Assignment #4 going out today
• Chris’s Ask Me Anything:
• On Zoom:
  – Chris AMA: July 15th, 11:30-12:30pm
def swap_elements_buggy(elem1, elem2):
    temp = elem1
    elem1 = elem2
    elem2 = temp

def main():
    my_list = [10, 20, 30]
    swap_elements_buggy(my_list[0], my_list[1])
    print(my_list)

Output: [10, 20, 30]
def swap_elements_working(alist, index1, index2):
    temp = alist[index1]
    alist[index1] = alist[index2]
    alist[index2] = temp

def main():
    my_list = [10, 20, 30]
    swap_elements_working(my_list, 0, 1)
    print(my_list)

Output: [20, 10, 30]
Learning Goals

1. Learning about slices
2. Working with 2-dimensional lists
Slices
What are Slices?

- Can cut up lists into "slices"
  - Slices are just sub-portions of lists
  - Slices are also lists themselves
  - Slicing creates a **new** list

**Example:**

```python
alist = ['a', 'b', 'c', 'd', 'e', 'f']
aslice = alist[2:4]
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>alist</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>aslice</td>
<td></td>
<td></td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>aslice</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are Slices?

• Can cut up lists into "slices"
  – Slices are just sub-portions of lists
  – Slices are also lists themselves
  – Slicing creates a **new** list

• Example:

```python
code
alist = ['a', 'b', 'c', 'd', 'e', 'f']
aslice = alist[2:4]
aslice[0] = 'x'
```

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>'b'</td>
<td>'c'</td>
<td>'d'</td>
<td>'e'</td>
<td>'f'</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
```

```python
code
aslice = alist[2:4]
aslice → ['x', 'd']
aslice[0] = 'x'
```

```python
code
alist = ['x', 'd']
```
General Form of Slice

• General form to get a slice

\[ \text{list}[\text{start}:\text{end}] \]

- Produces a new list with elements from \text{list} starting at index \text{start} up to (but not including) index \text{end}

• Example:

\text{alist} = ['a', 'b', 'c', 'd', 'e', 'f']

\[
\begin{array}{ccccccc}
\text{alist} & \rightarrow & 'a' & 'b' & 'c' & 'd' & 'e' & 'f' \\
& & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

alist[2:4] \rightarrow ['c', 'd']

alist[1:6] \rightarrow ['b', 'c', 'd', 'e', 'f']

alist[0:3] \rightarrow ['a', 'b', 'c']
I'll Take Another Slice!

• General form to get a slice

\[
list[start:end]
\]

– If \textit{start} is missing, default to use 0 in its place
– If \textit{end} is missing, default to use \texttt{len(list)} in its place
– Can also use negative indexes for \textit{start/end}

\[
alist \rightarrow \begin{array}{cccccc}
\text{'a'} & \text{'b'} & \text{'c'} & \text{'d'} & \text{'e'} & \text{'f'} \\
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

alist[2:-2] → ['c', 'd']
alist[-2:] → ['e', 'f']
alist[:−1] → ['a', 'b', 'c', 'd', 'e']
alist[:] → ['a', 'b', 'c', 'd', 'e', 'f']
Advanced Slices

- General form to get a slice, with a step

\[ \text{list} \left[ \text{start : end : step} \right] \]
- Take slice from \text{start} to \text{end}, progressing by \text{step}
- \text{step} can be negative (go backwards, so \text{start/\text{end}} are flipped)

\[
\begin{array}{cccccc}
-6 & -5 & -4 & -3 & -2 & -1 \\
\hline
\text{alist} & \text{'a'} & \text{'b'} & \text{'c'} & \text{'d'} & \text{'e'} & \text{'f'} \\
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[
\begin{align*}
\text{alist[1:5:2]} & \rightarrow \ [\text{'b'}, \text{'d'}] \\
\text{alist[::2]} & \rightarrow \ [\text{'a'}, \text{'c'}, \text{'e'}] \\
\text{alist[4:1:-1]} & \rightarrow \ [\text{'e'}, \text{'d'}, \text{'c'}] \quad \# \text{note start} \\
\text{alist[1:4:-1]} & \rightarrow \ [] \\
\text{alist[:::-1]} & \rightarrow \ [\text{'f'}, \text{'e'}, \text{'d'}, \text{'c'}, \text{'b'}, \text{'a'}] \\
\end{align*}
\]
Loops and Slices

• Can use for-each loop with slice
  – Slice is just a list, so you can use it just like a list
  – Recall loops with lists:

```python
for i in range(len(list)):
    # do something with list[i]

for elem in list:
    # do something with elem
```
Loops and Slices

• Can use for-each loop with slice
  – Slice is just a list, so you can use it just like a list
  – Now, for loops with slices (note: step is optional)

```python
for i in range(start, end, step):
    # do something with list[i]
```

```python
for elem in list[start:end:step]:
    # do something with elem
```

• Remember: if step is negative, then start should be greater than end
Deleting with Slices

• You can delete elements in a list with `del`

• Example:

```python
>>> num_list = [50, 30, 40, 60, 90, 80]
>>> del num_list[1]
>>> num_list
[50, 40, 60, 90, 80]
```

• Can use `del` with slice notation:

```python
>>> num_list = [50, 30, 40, 60, 90, 80]
>>> del num_list[1:4]
>>> num_list
[50, 90, 80]
```
Changing a List in Place

• Python provides some operations on whole list
  – These functions modify list in place (doesn't create new list)

• Function: `list.reverse()`
  – Reverses order of elements in the list

```python
>>> fun_list = [6, 3, 12, 4]
>>> fun_list.reverse()
>>> fun_list
[4, 12, 3, 6]
```

• Function: `list.sort()`
  – Sorts the elements of the list in increasing order

```python
>>> fun_list = [6, 3, 12, 4]
>>> fun_list.sort()
>>> fun_list
[3, 4, 6, 12]
```
2-Dimensional Lists
## 2-Dimensional List

- You can have a list of lists!
  - Each element of "outer" list is just another list
  - Can think of this like a grid

- Example:
  ```python
grid = [[1, 2], [3, 4], [5, 6]]
```

```
grid
  0 1 2
```

- Can be easier to think of like this:

```
grid
  0 1 2
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 2]</td>
<td>[3, 4]</td>
<td>[5, 6]</td>
<td></td>
</tr>
</tbody>
</table>
2-Dimensional List

- Um, can you zoom in on that…

```
[1, 2]   0
[3, 4]   1
[5, 6]   2
```

```
1  2
0  1
```

```
3  4
0  1
```

```
5  6
0  1
```
2-Dimensional List

- To access elements, specify index in "outer" list, then index in "inner" list

<table>
<thead>
<tr>
<th>grid</th>
<th>grid[0][0]</th>
<th>grid[0][1]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

grid[0][0] → 1
grid[1][0] → 3
grid[2][1] → 6
2-Dimensional List

- So what if I only specify one index?
  - grid[0] → [1, 2]
  - grid[1] → [3, 4]
  - grid[2] → [5, 6]

- Remember, grid is just a list of lists
  - Elements of "outer" list are just lists
Getting Funky With Lists

• Do the inner lists all have to be the same size?
  – No! Just be careful if they are not.
    
    \[
    \text{jagged} = \begin{bmatrix}
    [1, 2, 3], & [4], & [5, 6] \\
    \end{bmatrix}
    \]
    
    \[
    \text{jagged}[0] \rightarrow [1, 2, 3] \\
    \text{jagged}[1] \rightarrow [4] \\
    \text{jagged}[2] \rightarrow [5, 6]
    \]

• Can I have more than two dimensions?
  – Sure! You can have as many as you like (within reason).
    
    \[
    \text{cube} = \begin{bmatrix}
    \begin{bmatrix}
    [1, 2], & [3, 4] \\
    \end{bmatrix}, & \begin{bmatrix}
    [5, 6], & [7, 8] \\
    \end{bmatrix} \\
    \end{bmatrix}
    \]
    
    \[
    \text{cube}[0] \rightarrow \begin{bmatrix}
    [1, 2], & [3, 4] \\
    \end{bmatrix} \\
    \text{cube}[0][1] \rightarrow [3, 4] \\
    \text{cube}[0][1][0] \rightarrow 3
    \]
def swap(grid, row1, col1, row2, col2):
    temp = grid[row1][col1]
    grid[row1][col1] = grid[row2][col2]
    grid[row2][col2] = temp

def main():
    my_grid = [[10, 20, 30], [40, 50, 60]]
    swap (my_grid, 0, 1, 1, 2)
    print(my_grid)

Output: 

[[10, 60, 30], [40, 50, 20]]
def main():
    grid = [[10, 20], [40], [70, 80, 100]]
    rows = len(grid)
    for row in range(rows):
        cols = len(grid[row])
        for col in range(cols):
            print(f"grid[{row}][{col}] = {grid[row][col]}")

Output:

grid[0][0] = 10
grid[0][1] = 20
grid[1][0] = 40
grid[2][0] = 70
grid[2][1] = 80
grid[2][2] = 100
def main():
    grid = [[1, 2], [10, 11], [20, 21]]
    rows = len(grid)
    cols = len(grid[0])
    for row in range(rows):
        for col in range(cols):
            print(f"grid[{row}][{col}] = {grid[row][col]}")

Output:

grid[0][0] = 1
grid[0][1] = 2
grid[1][0] = 10
grid[1][1] = 11
grid[2][0] = 20
grid[2][1] = 21
def main():
    grid = [[10, 20], [40], [70, 80, 100]]
    for row in grid:
        for col in row:
            print(col)

Output:

10
20
40
70
80
100
def create_grid(rows, cols, value):
    grid = []  # Create empty grid
    for y in range(rows):  # Make rows one by one
        row = []
        for x in range(cols):  # Build up each row
            row.append(value)  # by appending to list
        grid.append(row)  # Append row (list) # onto grid
    return grid

Console:

>>> create_grid(2, 4, 1)
[[1, 1, 1, 1], [1, 1, 1, 1]]
>>> create_grid(3, 2, 5)
[[5, 5], [5, 5], [5, 5]]
Putting it all together: tictactoe.py

(This program give you practice with a lot of concepts!)
Learning Goals

1. Learning about slices
2. Working with 2-dimensional lists