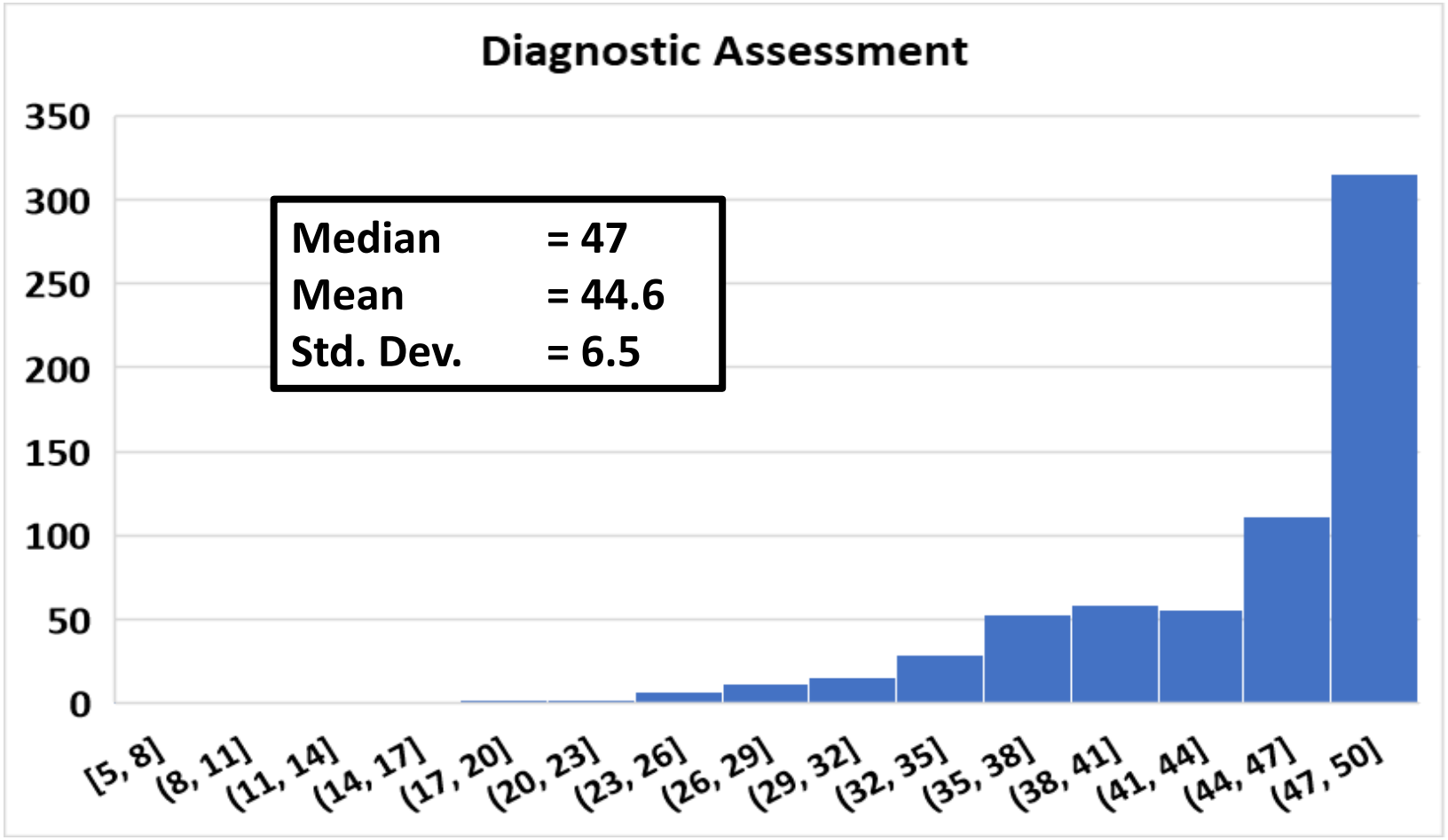
More Lists

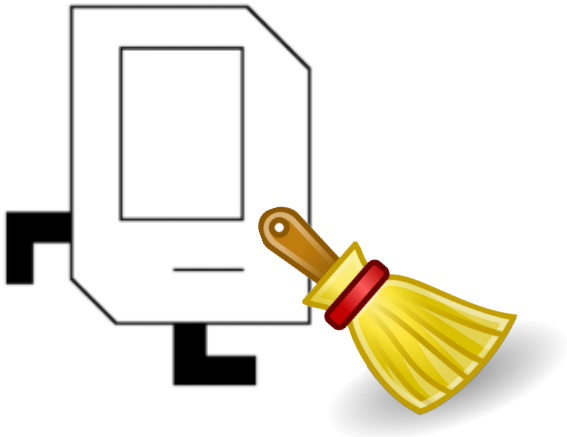
Chris Piech and Mehran Sahami
CS106A, Stanford University

Diagnostic Assessment



Regrade requests should be made on Gradescope between May 6th at 12pm (PDT) and Sun., May 10th at 12pm (PDT)

Housekeeping



- Assignment #3 due today
- Assignment #4 going out tonight
- Pixar Night: Thursday May 7th, 4:30-6pm PDT on Zoom
- Chris and Mehran holding Ask Me Anything (AMA) sessions
  - On class Ed Forum (or maybe Zoom)
  - Chris AMA: May 9 at 9am-10am (PDT)
  - Mehran AMA: May 14 at 4pm-5pm (PDT)

Swapping Elements in a List - Sad

```
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]`

Swapping Elements in a List - Happy

```
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:

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
```

alist →

'a'	'b'	'c'	'd'	'e'	'f'
0	1	2	3	4	5

```
aslice = alist[2:4]
```

aslice →

'c'	'd'
0	1

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:

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
```

alist →

'a'	'b'	'c'	'd'	'e'	'f'
0	1	2	3	4	5

```
aslice = alist[2:4]
```

aslice →

'x'	'd'
0	1

```
aslice[0] = 'x'
```

General Form of Slice

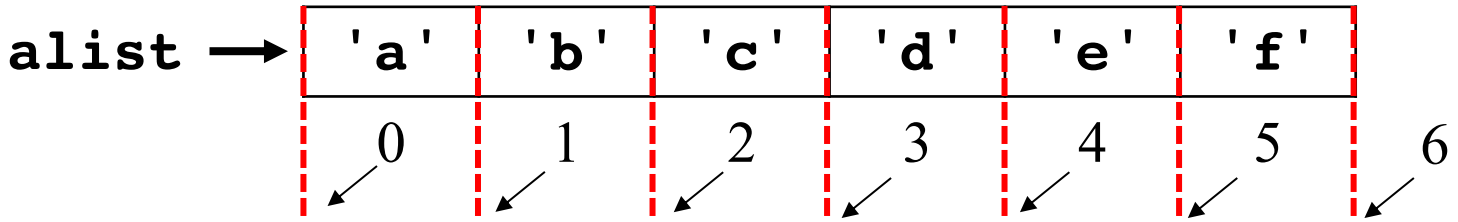
- General form to get a slice

  ***list*[*start*:*end*]**

  – Produces a new list with elements from ***list*** starting at index ***start*** up to (but not including) index ***end***

- Example:

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
```



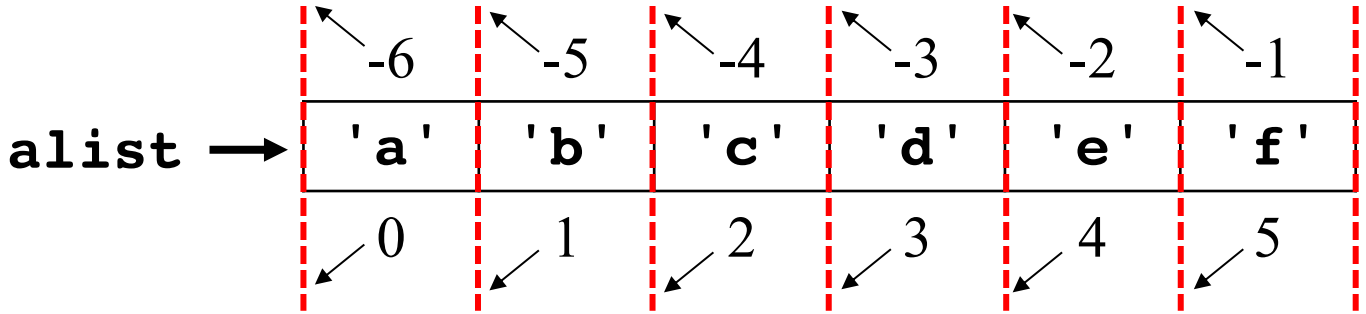

```
alist[2:4]   →   ['c', 'd']
alist[1:6]   →   ['b', 'c', 'd', 'e', 'f']
alist[0:3]   →   ['a', 'b', 'c']
```

I'll Take Another Slice!

- General form to get a slice

  ***list*[*start*:*end*]**

  - If ***start*** is missing, default to use 0 in its place
  - If ***end*** is missing, default to use **`len(`*list*`)`** in its place
  - Can also use negative indexes for ***start*/*end***




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

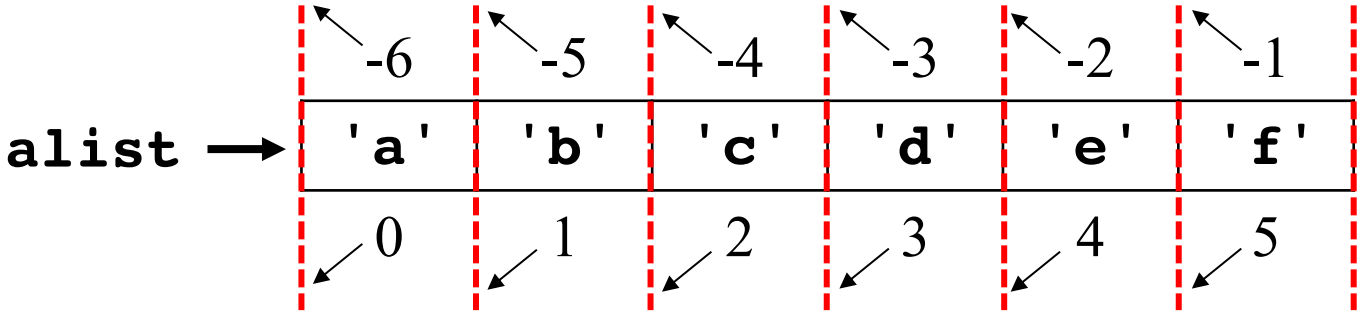
Advanced Slices

- General form to get a slice, with a step

  ***list*[*start*:*end*:*step*]**

  – Take slice from ***start*** to ***end***, progressing by ***step***

  – ***step*** can be negative (go backwards, so ***start*/*end*** are flipped)




```
alist[1:5:2]  →  ['b', 'd']
alist[::2]    →  ['a', 'c', 'e']
alist[4:1:-1] →  ['e', 'd', 'c']  # note start
alist[1:4:-1] →  []
alist[::-1]   →  ['f', 'e', 'd', 'c', 'b', 'a']
```

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:

```
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)

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

```
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:

```
>>> 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:

```
>>> 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

```
>>> 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

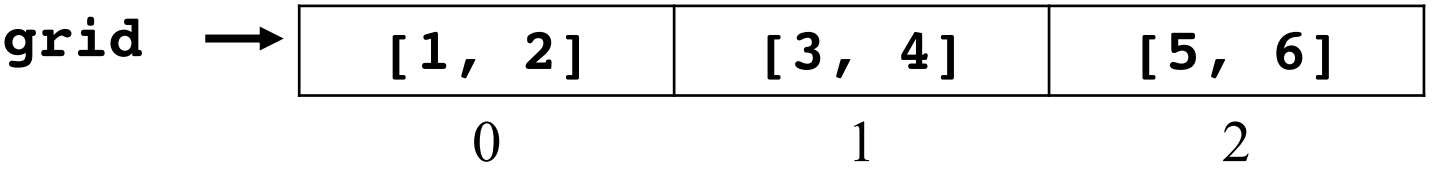
```
>>> fun_list = [6, 3, 12, 4]
>>> fun_list.sort()
>>> fun_list
[3, 4, 6, 12]
```


2-Dimensional Lists

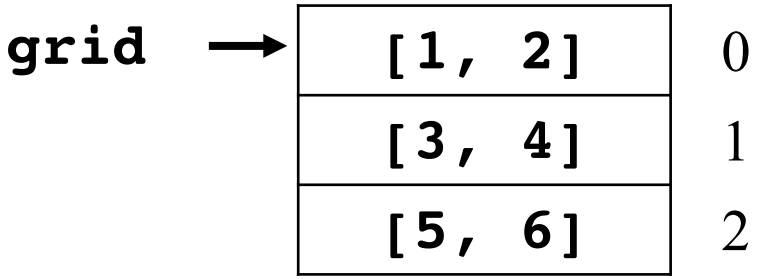
2-Dimensional List

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

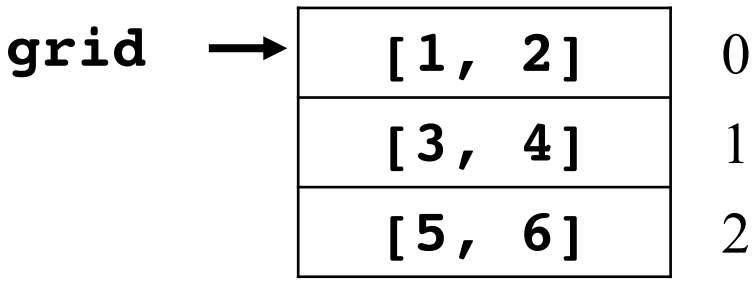
```
grid = [[1, 2], [3, 4], [5, 6]]
```



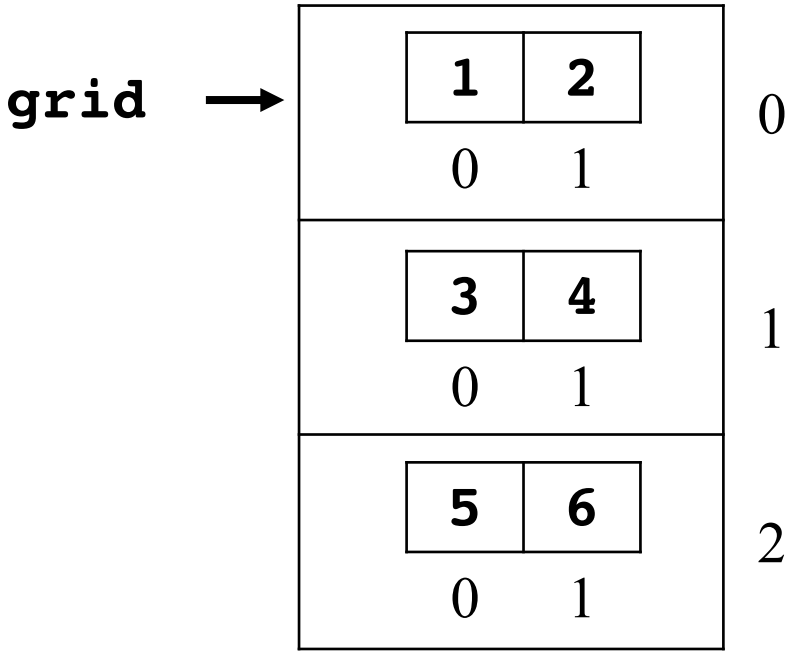
- Can be easier to think of like this:



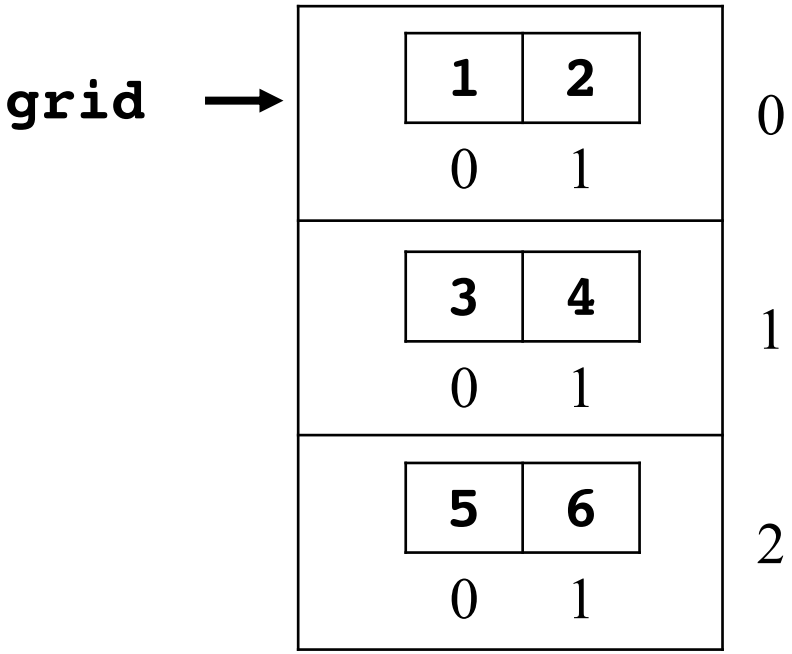
2-Dimensional List



- Um, can you zoom in on that…



2-Dimensional List

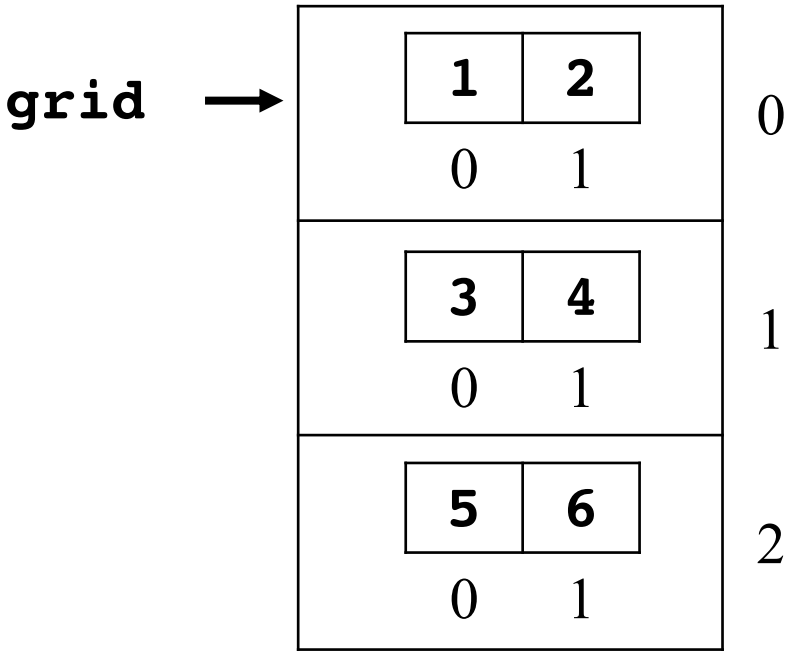


grid[0][0] 1	grid[0][1] 2
grid[1][0] 3	grid[1][1] 4
grid[2][0] 5	grid[2][1] 6

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

```
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.

```
jagged = [[1, 2, 3], [4], [5, 6]]
jagged[0]      →  [1, 2, 3]
jagged[1]      →  [4]
jagged[2]      →  [5, 6]
```

- Can I have more than two dimensions?
  - Sure! You can have as many as you like (within reason).

```
cube = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
cube[0]        →  [[1, 2], [3, 4]]
cube[0][1]     →  [3, 4]
cube[0][1][0]  →  3
```


Swapping Elements in a Grid

```
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]]`

Looping Through a List of Lists

```
def main():
    grid = [[10, 20], [40], [70, 80, 100]]
    rows = len(grid)
    for i in range(rows):
        cols = len(grid[i])
        for j in range(cols):
            print("grid[" + str(i) + "][" + str(j)
                  + "] = " + str(grid[i][j]))
```

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
```


Simplified With a True Grid

```
def main():
    grid = [[1, 2], [10, 11], [20, 21]]
    rows = len(grid)
    cols = len(grid[0])
    for i in range(rows):
        for j in range(cols):
            print("grid[" + str(i) + "][" + str(j)
                  + "] = " + str(grid[i][j]))
```

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
```


Using For-Each With 2-D List

```
def main():
    grid = [[10, 20], [40], [70, 80, 100]]
    for row in grid:
        for elem in row:
            print(elem)
```

Output:

```
10
20
40
70
80
100
```


Creating a 2-D List

```
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