Review:
Lists as parameters

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

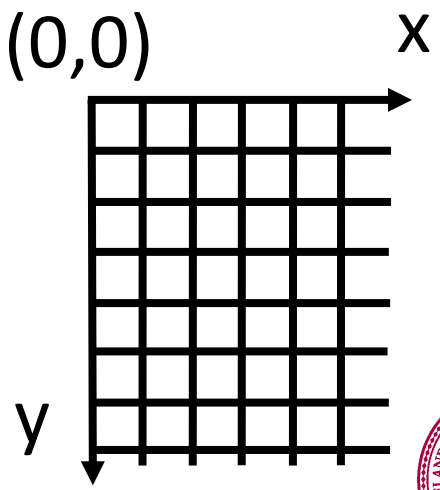
```
def swap_elements_working(alist, index1, index2):
    temp = alist[index1]
    alist[index1] = alist[index2]
    alist[index2] = temp


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

Output: `[20, 10, 30]`

More fun with images!
Mirroring an image

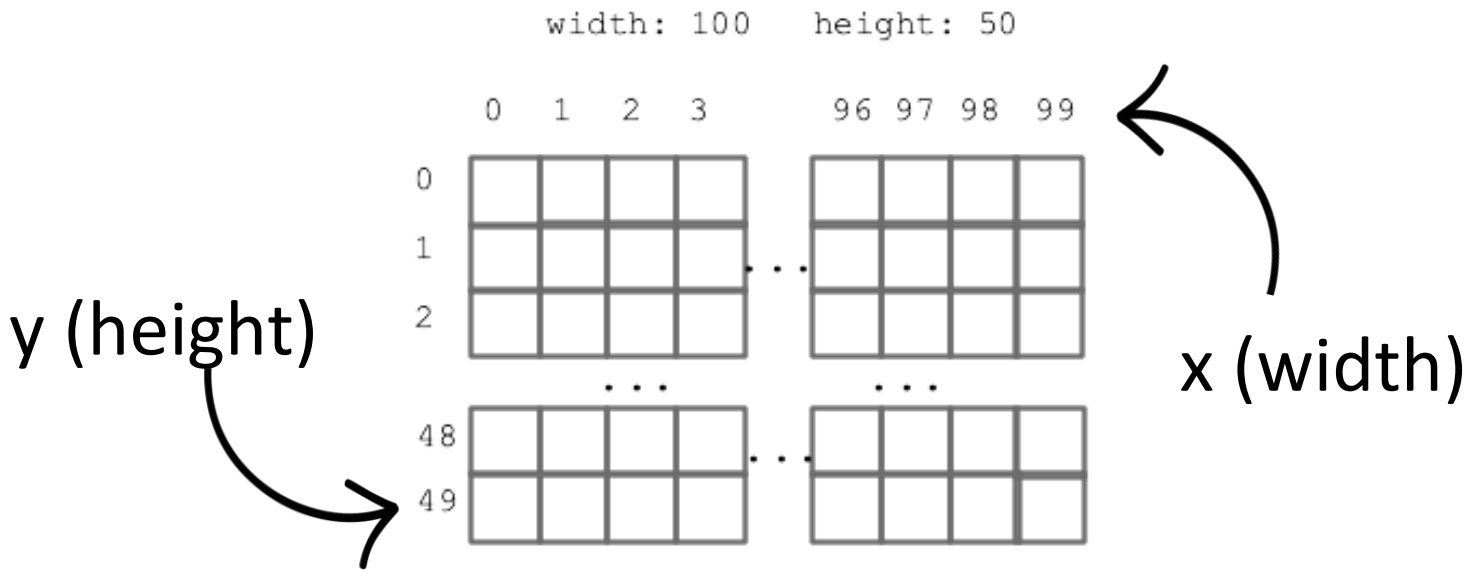
Recall, Images

- Image made of square pixels
  - Example: flower.png
- Each pixel has x and y coordinates in the image
  - The origin (0, 0) is at the upper-left corner
  - y **increases** going down, x increases going right
- Each pixel has single color encoded as 3 **RGB** values
  - R = red; G = green; B = blue
  - Each value represents brightness for that color (red, green, or blue)
  - Can set RGB values to make any color!



Nested Loops

```
image = SimpleImage(filename)
width = image.width
height = image.height

for y in range(height):
    for x in range(width):
        pixel = image.get_pixel(x, y)
        # do something with pixel
```



Mirroring an Image

```
def mirror_image(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height

    # Create new image to contain mirror reflection
    mirror = SimpleImage.blank(width * 2, height)

    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            mirror.set_pixel(x, y, pixel)
            mirror.set_pixel((width * 2) - (x + 1), y, pixel)
    return mirror
```

I wanna see it!

What's The Difference?

```
def darker(filename):
    img = SimpleImage(filename)
    for px in img:
        px.red = px.red // 2
        px.green = px.green // 2
        px.blue = px.blue // 2
    return img
```

```
def darker(filename):
    img = SimpleImage(filename)
    for y in range(img.height):
        for x in range(img.width):
            px = img.get_pixel(x, y)
            px.red = px.red // 2
            px.green = px.green // 2
            px.blue = px.blue //  2
    return img
```

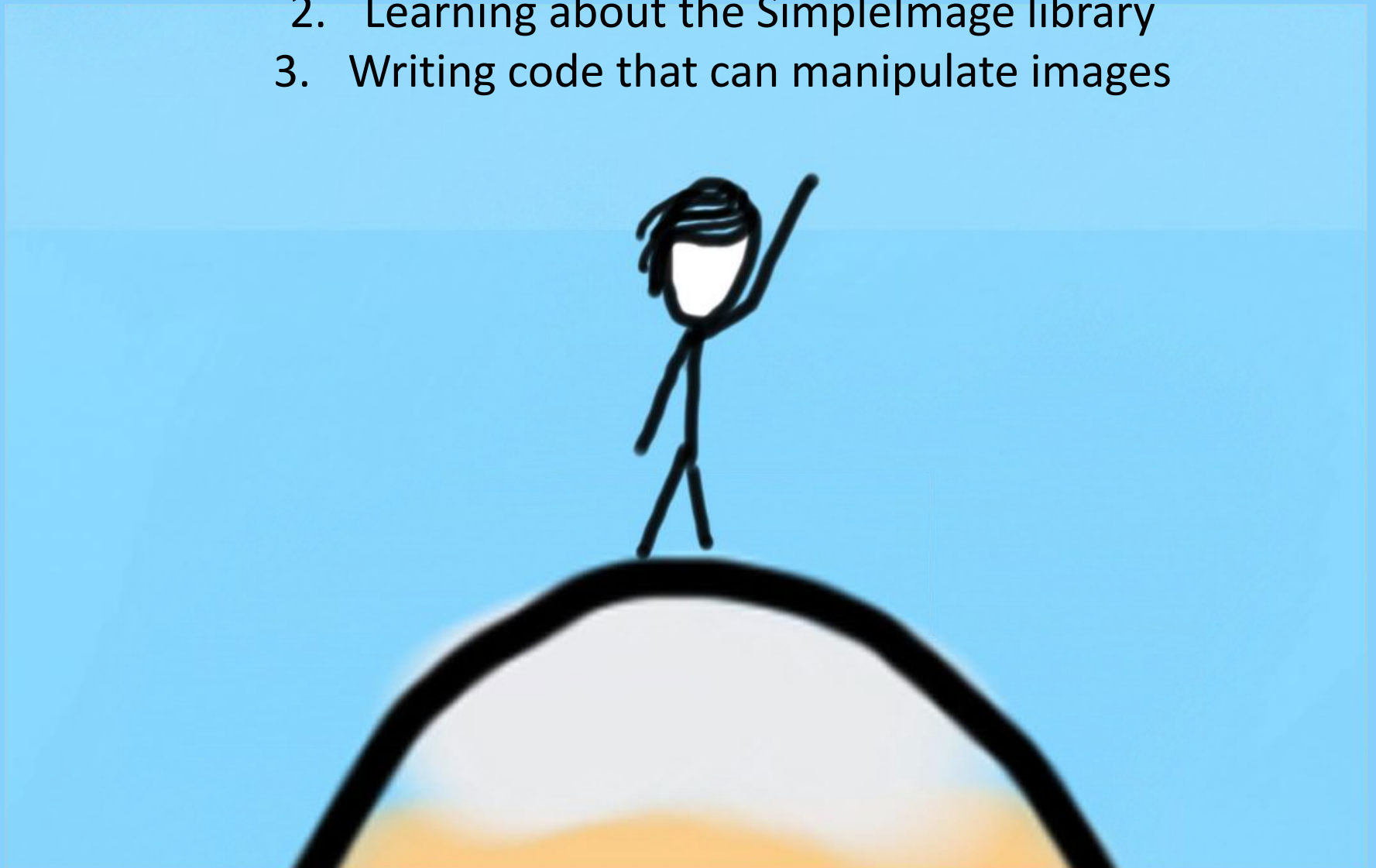
Nothing!

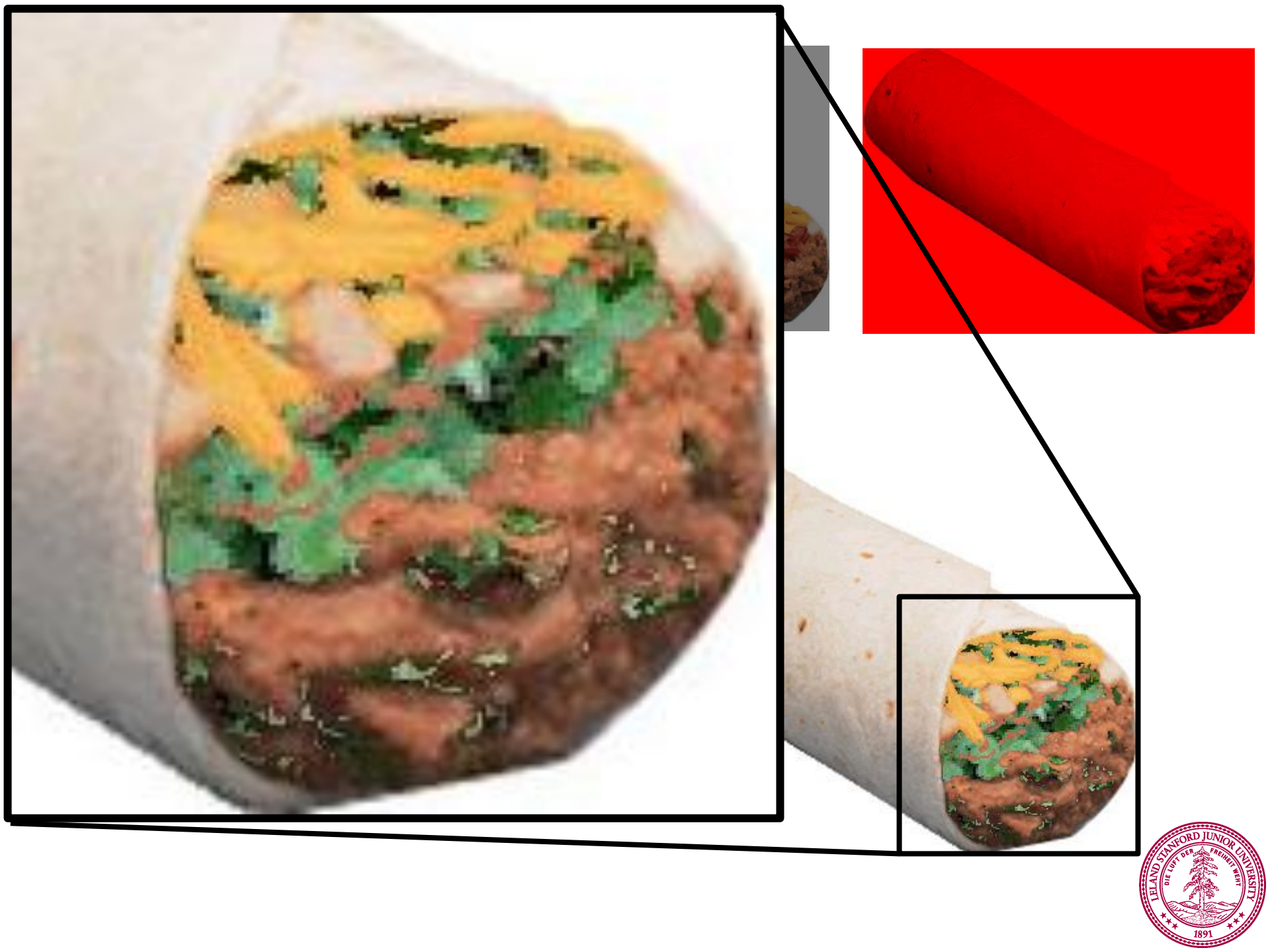
We only want to use nested for loops if we care about **x** and **y**.

(Needed that for mirroring image.)

Learning Goals

1. Understanding how images are represented
2. Learning about the SimpleImage library
3. Writing code that can manipulate images





What are the ethics of this?

Welcome: Dr. Katie Creel

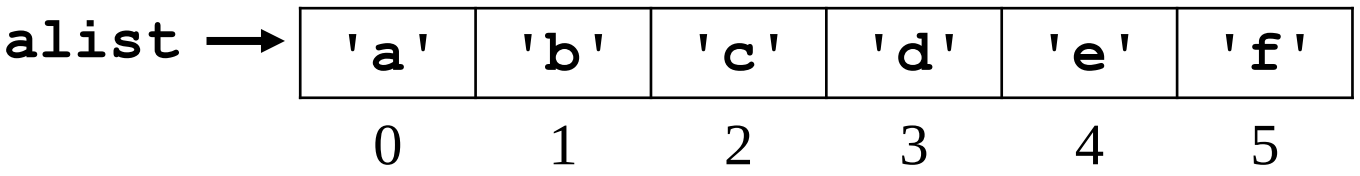
Learning Goals

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

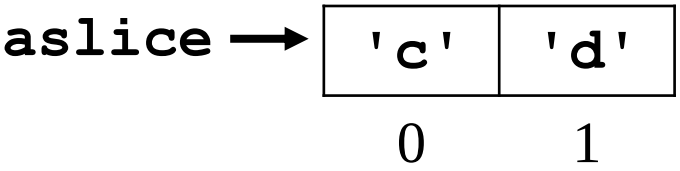
What are Slices?

- 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']
```




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



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

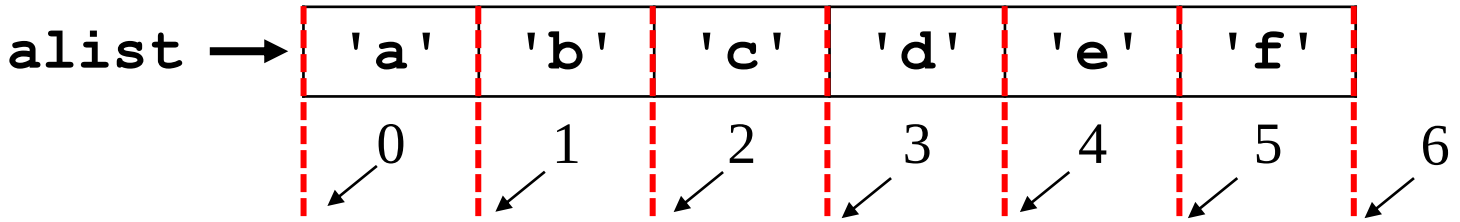
- General form to get a slice

  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']
```



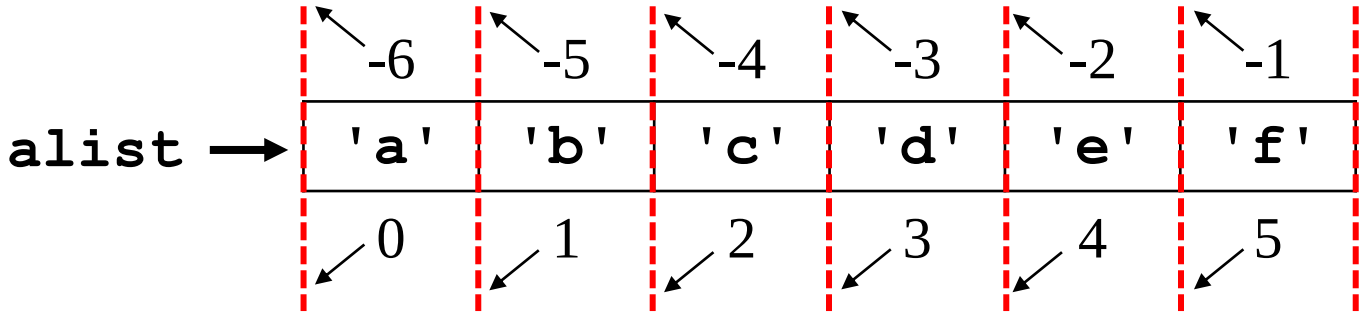

```
alist[2:4]   →   ['c', 'd']
alist[1:6]   →   ['b', 'c', 'd', 'e', 'f']
alist[0:3]   →   ['a', 'b', 'c']
```

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']
```

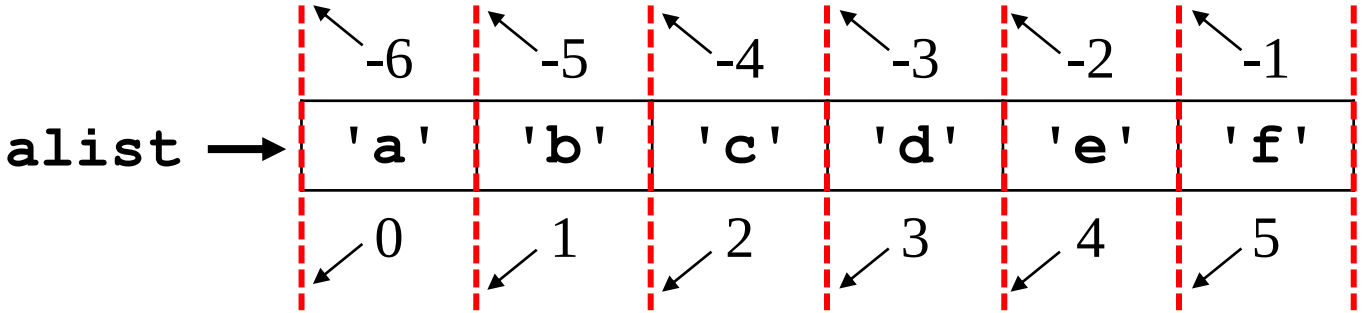
Advanced Slices

- 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

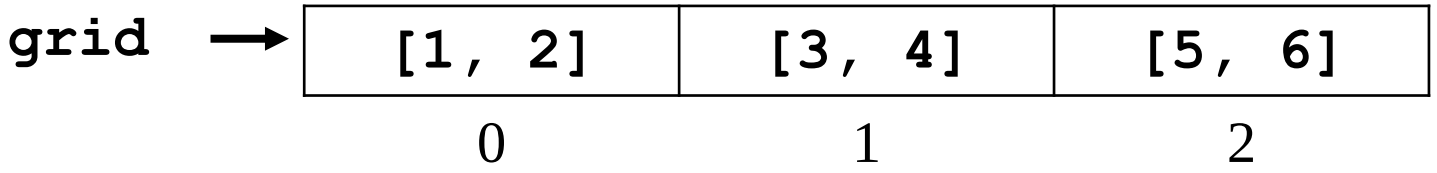
```
>>> fun_list = [6, 3, 12, 4]
>>> fun_list.sort()
>>> fun_list
[3, 4, 6, 12]
```

2-Dimensional Lists

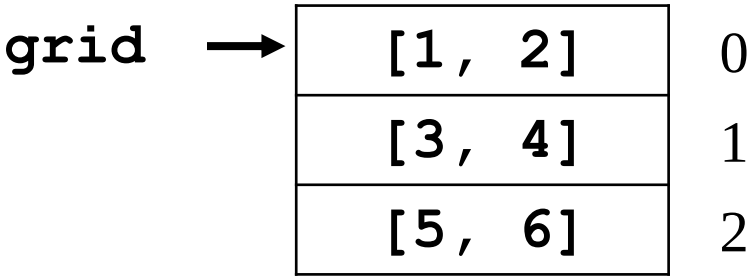
2-Dimensional List

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

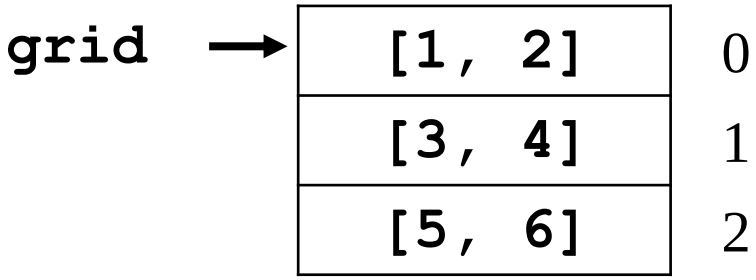
```
grid = [[1, 2], [3, 4], [5, 6]]
```



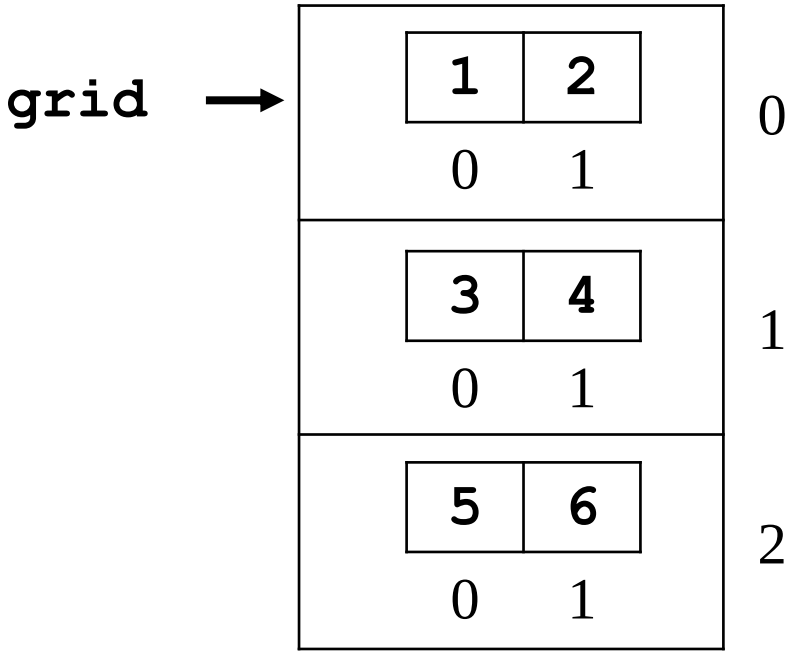
- Can be easier to think of like this:



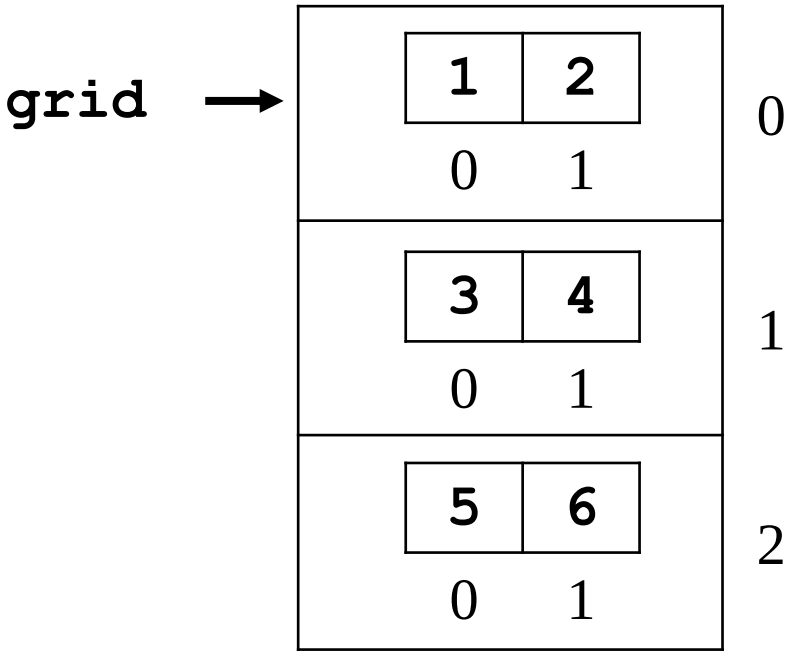
2-Dimensional List



- 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
```