More Lists

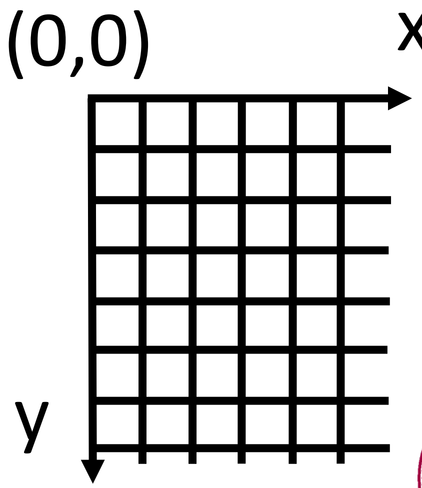
Chris Gregg and Mehran Sahami
CS106A, Stanford University

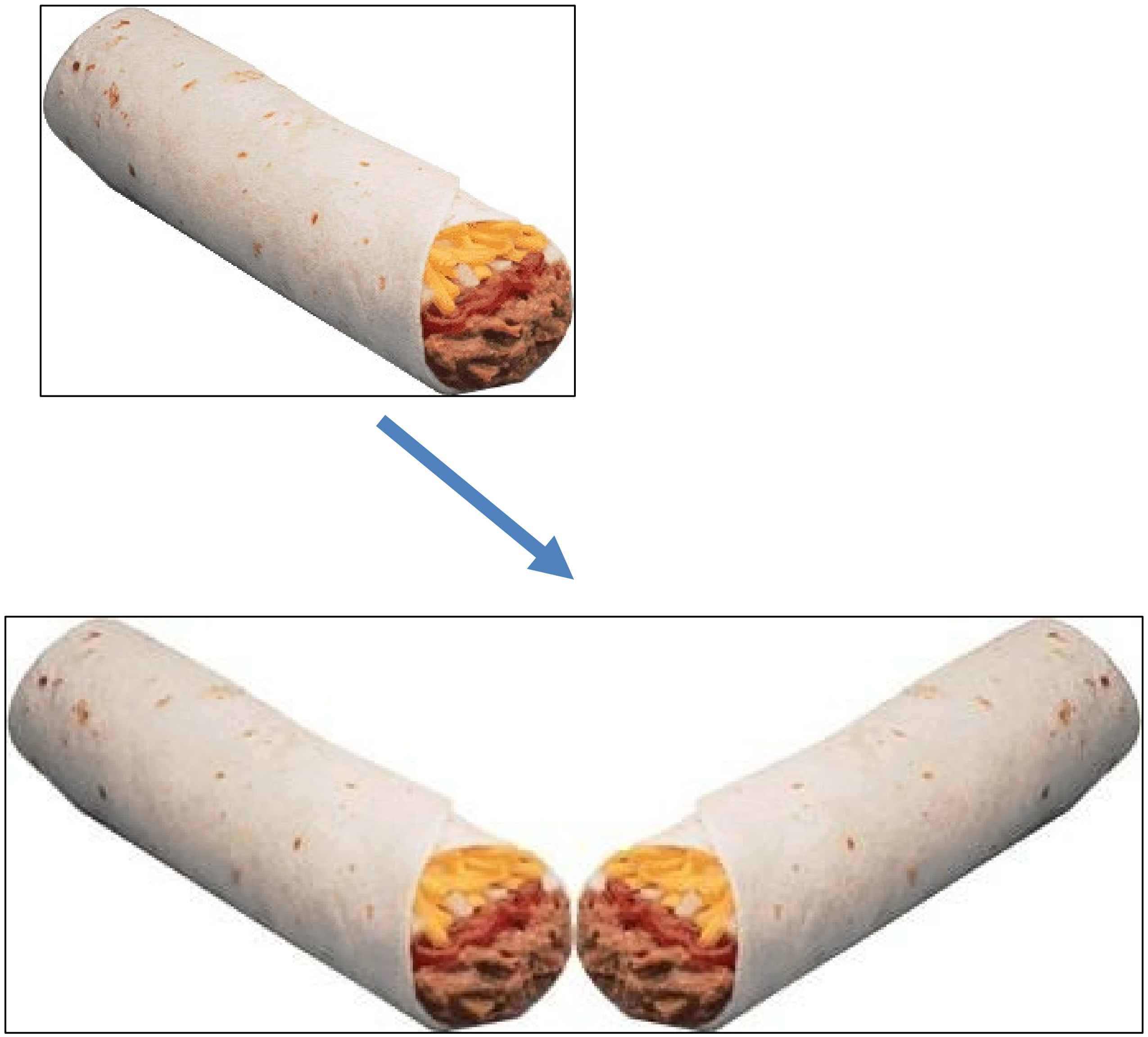
But first...
More fun with images!
Mirroring an image

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 **<u>increases</u>** 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!

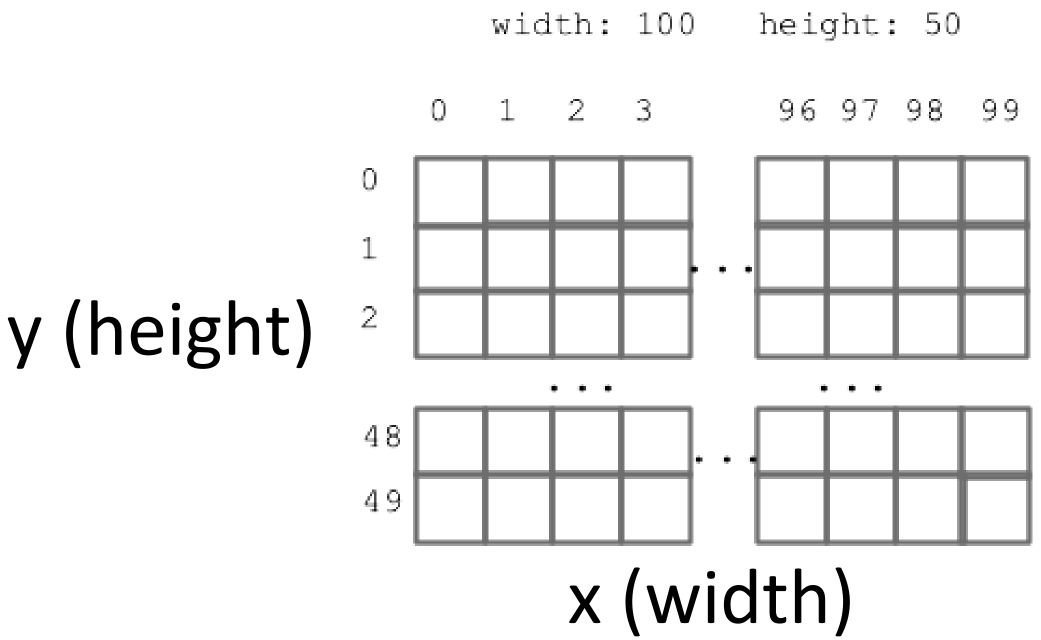




Nested Loops

```
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. Learning about more about lists
2. Learning about slices
3. Working with 2-dimensional lists

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)
```

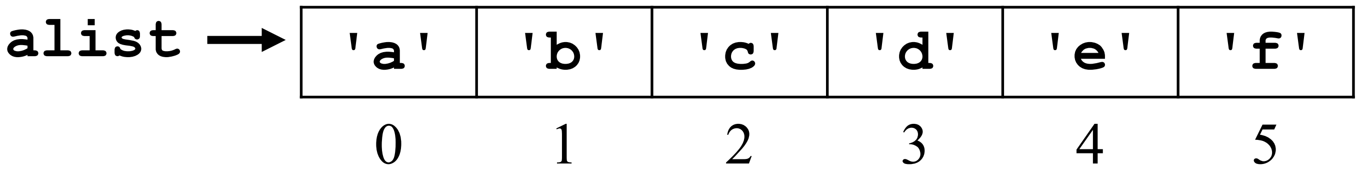
Output:

```
[20, 10, 30]
```

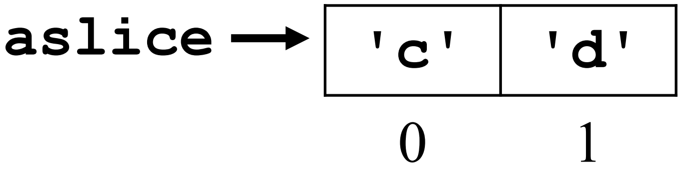
What are Slices?

- 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]
```

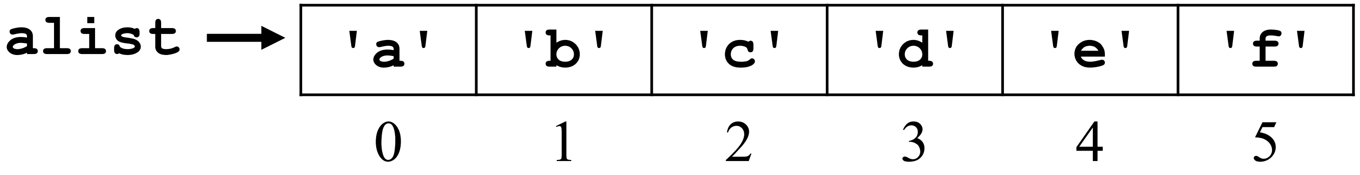


What are Slices?

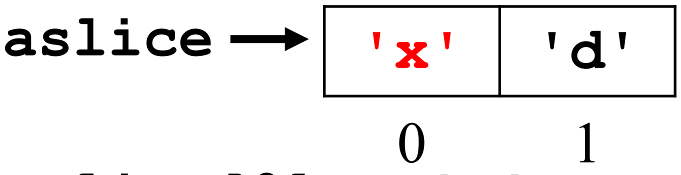
- Can cut up lists into "slices"
  - Slices are just sub-portions of lists
  - Slices are also lists themselves
  - Slicing creates a **new** list

- Example:

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



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



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

General Form of Slice

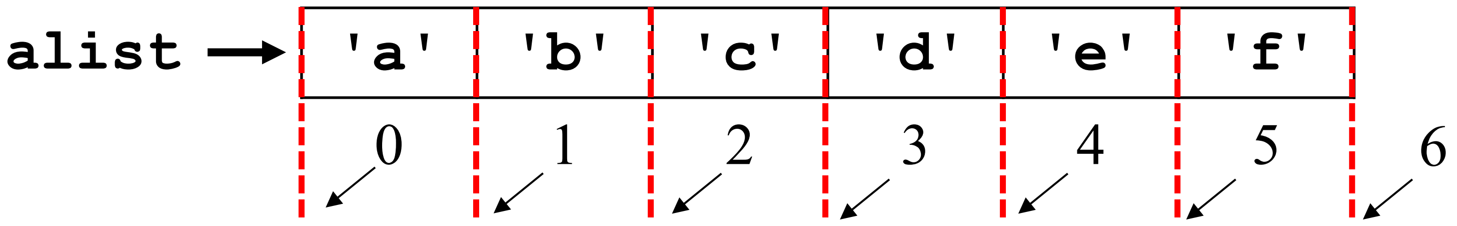
- General form to get a slice

  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']
```

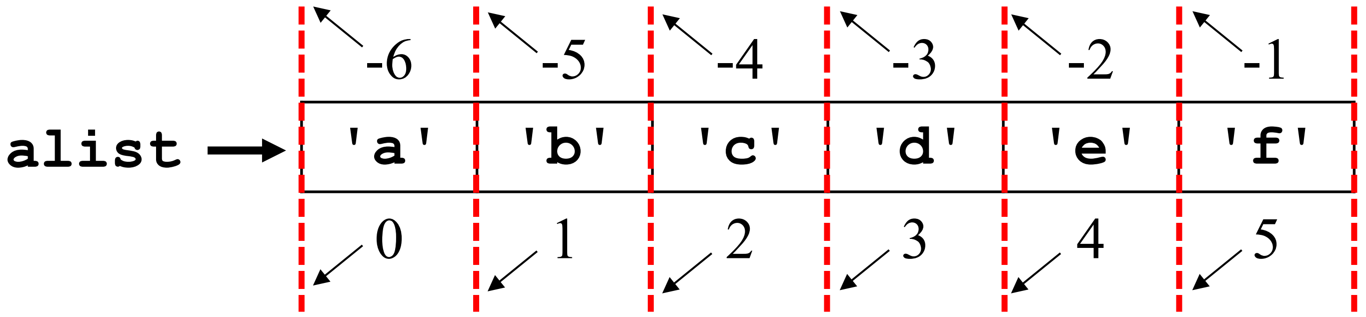


```
alist[2:4]    →   ['c', 'd']
alist[1:6]    →   ['b', 'c', 'd', 'e', 'f']
alist[0:3]    →   ['a', 'b', 'c']
```

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']
```

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()
print(fun_list)
```

Printed on terminal: `[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()
print(fun_list)
```

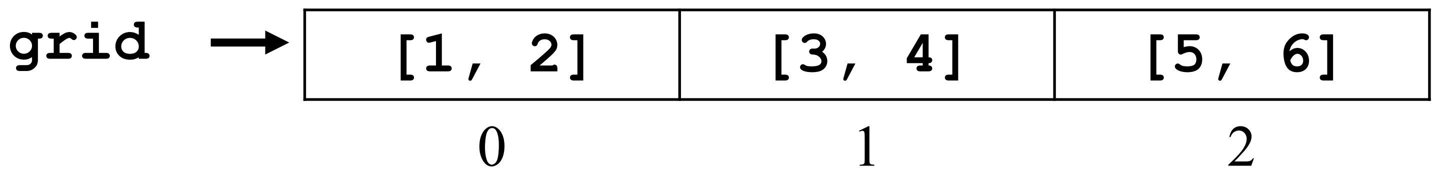
Printed on terminal: `[3, 4, 6, 12]`

2-Dimensional Lists

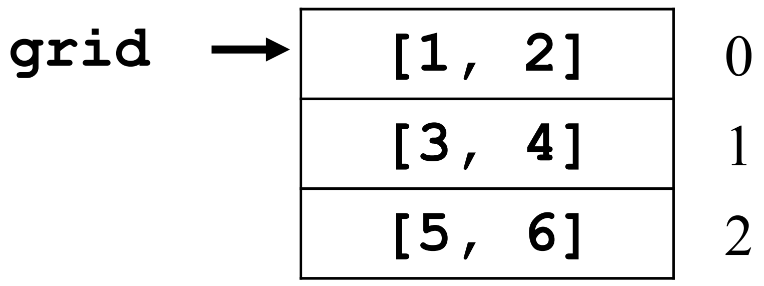
2-Dimensional List

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

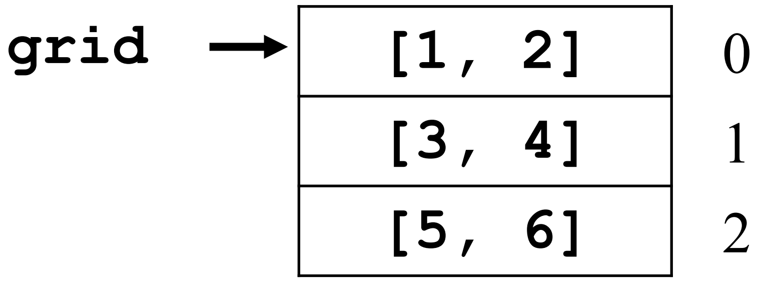
```
grid = [[1, 2], [3, 4], [5, 6]]
```



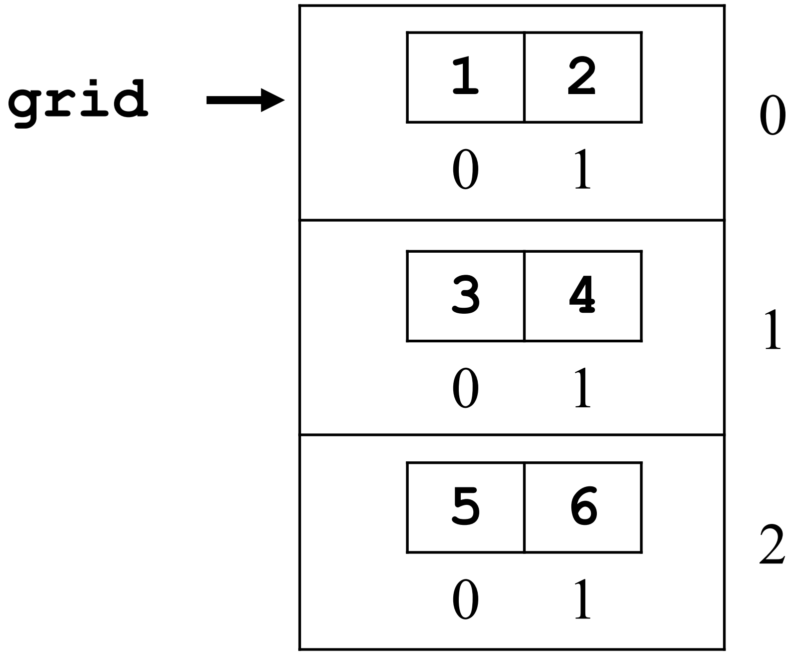
- Can be easier to think of like this:



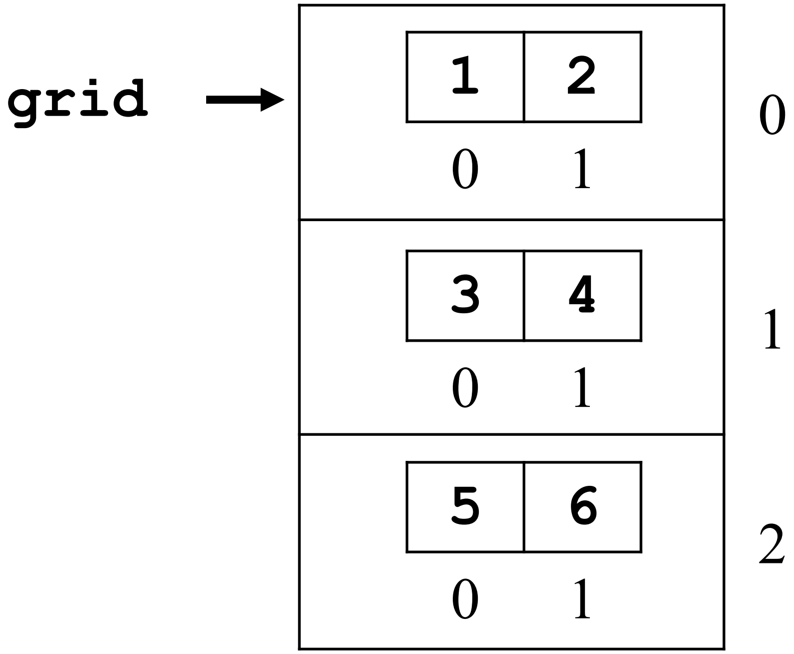
2-Dimensional List



- Um, can you zoom in on that…



2-Dimensional List

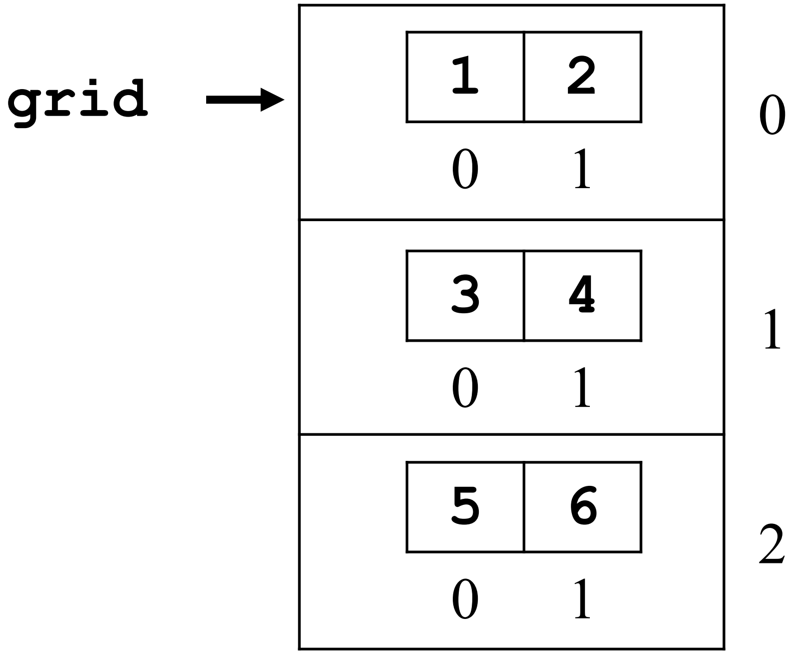


`grid[0][0]` 1	`grid[0][1]` 2
`grid[1][0]` 3	`grid[1][1]` 4
`grid[2][0]` 5	`grid[2][1]` 6

- 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

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]]
```

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]
```

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(f"grid[{i}][{j}] = {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)
    for i in range(rows):
        cols = len(grid[0])
        for j in range(cols):
            print(f"grid[{i}][{j}] = {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
```

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(f"grid[{i}][{j}] = {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
```

Learning Goals

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