CMSC201
Computer Science I for Majors

Lecture 09 – While Loops

Prof. Jeremy Dixon

Last Class We Covered

- Using **`for`** loops
  - Syntax
  - Using it to iterate over a list
  - Using it for "counting" the number of actions
- The **`range()`** function
  - Syntax
  - Three forms: one, two, or three numbers

Any Questions from Last Time?

Today's Objectives

- To learn about and use a **`while`** loop
  - To understand the syntax of a **`while`** loop
  - To use a **`while`** loop for interactive loops
- To learn two different ways to mutate a list
  - **`append()`** and **`remove()`**
- To apply our knowledge to created nested loops
- To touch (briefly) on two-dimensional lists

Review: Looping

Remember our Average `for` Loop?

- Use a `for` loop to find the average from a list of numbers

```
nums  = [98, 75, 89, 100, 45, 82]
total = 0   # we have to initialize total to zero

for n in nums:
    total = total + n  # so that we can use it here
avg = total / len(nums)
print("Your average in the class is: ", avg)
```

Getting Flexible Input

- What if we only want positive numbers?
- And we want to re-prompt the user if they enter a negative number?
  - And *keep* re-prompting until they enter a positive
- We can't do this with a `for` loop – why?
  - They only run a pre-set number of times
  - We don't know how many times to re-prompt

Looping

- Python has two kinds of loops, and they are used for two different purposes

- The **`for`** loop:
  - Good for *iterating* over a list
  - Good for counted iterations
- The **`while`** loop
  - Good for almost everything else

what we're covering today

`while` Loops: Syntax and Uses

The `while` Loop

- The **`while`** loop is used when we're not
  - Iterating over a list
  - Doing a "counted" loop

- Works the way its name implies:

  While a certain condition is not yet met:

  Continue to repeatedly do a thing

Parts of a `while` Loop

- Here's some example code… let's break it down

```
date = 0

while date < 1 or date > 31:
    date = int(input("Enter the day: "))

print("Today is September", date)
```

Parts of a `while` Loop

- Here's some example code… let's break it down

initialize the variable the `while` loop will use for its decision

```
date = 0

while date < 1 or date > 31
    date = int(input("Enter the day: "))

print("Today is September",
```

the loop's Boolean condition (loop runs until this is `False`)

the body of the loop (must change the value of the loop variable)

How a `while` Loop Works

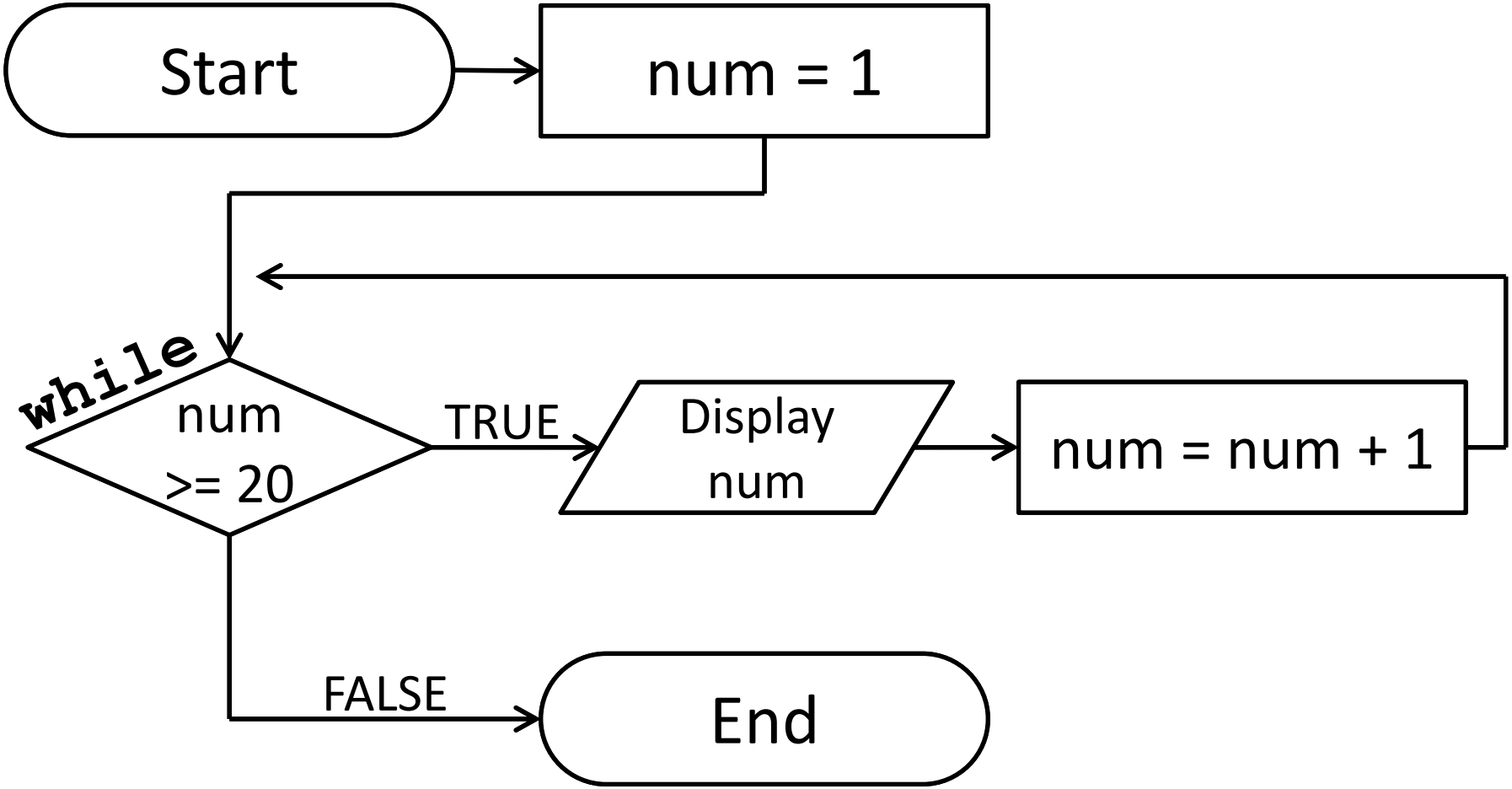
- The **`while`** loop requires a Boolean condition
  - That it then evaluates to either **`True`** or **`False`**
- If the condition is **`True`**:
  - Body of **`while`** loop is executed
- If the condition is **`False`**:
  - Body of **`while`** loop is skipped

Example `while` Loop

- We can use a **`while`** loop to do a "counting" loop, just like we used a **`for`** loop

```
num  = 1          # we have to initialize num to zero

while num <= 20:          # so that we can use it here
    print(num)
    num = num + 1         # change the loop variable
```

Example `while` Loop



Infinite Loops and Other Problems

Infinite Loops

- An *infinite loop* is a loop that will run forever

- Can we have an infinite loop using **`for`**?
  - No! – the **`for`** loop goes through a set number of steps (iterating or counting) and will always end
- Can we have an infinite loop using **`while`**?
  - Yes! – the **`while`** loop's loop variable is controlled by us, and we can make mistakes

Infinite Loop Example #1

- Why doesn't this loop end? What will fix it?

```
age = 0
while age < 18:    # can't vote until 18
    print("You can't vote at age", age)

print("Now you can vote! Yay!")
```

Control+C to break an infinite loop in Python!

Infinite Loop Example #1

- Why doesn't this loop end? What will fix it?

the loop variable (**age**) never changes, so the condition will never evaluate to **True**

```
age = 0
while age < 18:    # can't vote until 18
    print("You can't vote at age", age)

print("Now you can vote! Yay!")
```

Infinite Loop Example #2

- Why doesn't this loop end? What will fix it?

```
while True:
    # ask user for name
    name = input("What is your name? ")

print("Hello", name + "!")
```

Infinite Loop Example #2

- Why doesn't this loop end? What will fix it?

`True` will never evaluate to `False`, so the loop will never exit

```
while True:
    # ask user for name
    name = input("What is your name? ")

print("Hello", name + "!")
```

Infinite Loop Example #3

- Why doesn't this loop end? What will fix it?

```
cookiesLeft = 50

while cookiesLeft > 0:
    # eat a cookie
    cookiesLeft = cookiesLeft + 1

print("No more cookies!")
```

Infinite Loop Example #3

- Why doesn't this loop end? What will fix it?

```
cookiesLeft = 50

while cookiesLeft > 0:
    # eat a cookie
    cookiesLeft = cookiesLeft + 1

print("No more cookies!")
```

the loop body is INCREASING the number of cookies, so we'll never reach zero!

Loop Body Isn't Being Run

- Unlike most **`for`** loops, a **`while`** loop's body may be skipped over entirely
  - If the Boolean condition is initially **`False`**

```
militaryTime = 1300

while (militaryTime < 1200):
    print("Good morning!")
    militaryTime = militaryTime + 1
```

Updating and Changing Lists

Mutating Lists

- Remember that lists are defined as "mutable sequences of arbitrary objects"
  - "Mutable" just means we can change them

- So far, the only thing we've been able to change about our lists are their contents
  - But we can also change their size, by adding and removing elements

Two List Functions

- There are two functions we'll cover today that can add and remove things to our lists

```
append()
remove()
```

- There are more, but we'll cover them later

List Function: `append()`

- The **`append()`** function lets us add items to the end of a list, increasing its size

  `LISTNAME.append(ITEM_TO_APPEND)`

- Useful for creating a list from flexible input
  - Allows the list to expand as the user needs
  - No longer need to initialize lists to **`[None]*NUM`**

Example of `append()`

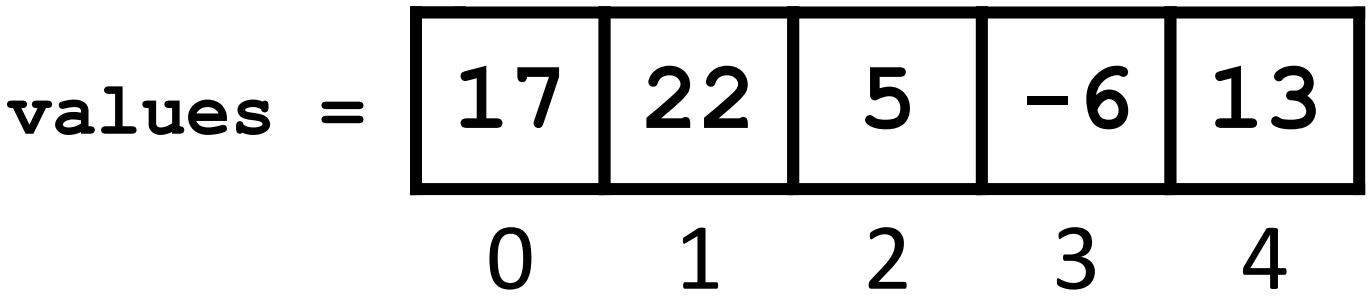
- We can use `append()` to create a list of numbers (continuing until the user enters 0)

```
values = []    # initialize the list to be empty
userVal = 1    # give our loop variable a value

while userVal != 0:
    userVal = int(input("Enter a number, 0 to stop"))
    if userVal != 0:          # only append if it's valid
        values.append(userVal) # add value to the list
```

Example of `append()`

- We can use **`append()`** to create a list of numbers (continuing until the user enters 0)

```
while userVal != 0:
    userVal = int(input("Enter a number, 0 to stop"))
    if userVal != 0:          # only append if it's valid
        values.append(userVal) # add value to the list
```



List Function: `remove()`

- The `remove()` function lets us remove an item from the list – specifically, it finds and removes the first instance of a given value

```
LISTNAME.remove(ITEM_TO_REMOVE)
```

- Useful for deleting things that no longer matter
  - For example, removing students who have dropped the class from the class roster
  - Keeps the list from having empty elements

Example of `remove()`

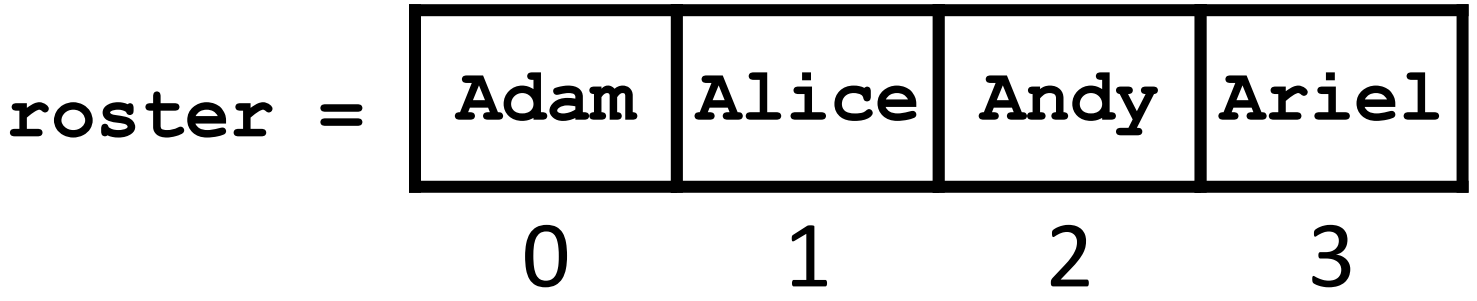
- We can use `remove()` to remove students who have dropped the class from the roster

```
roster = ["Adam", "Alice", "Andy", "Ariel"]

roster.remove("Adam")    # Adam has dropped the class

# Bob is not in roster, so this causes an error
roster.remove("Bob")
```

Example of `remove()`

- We can use `remove()` to remove students who have dropped the class from the roster

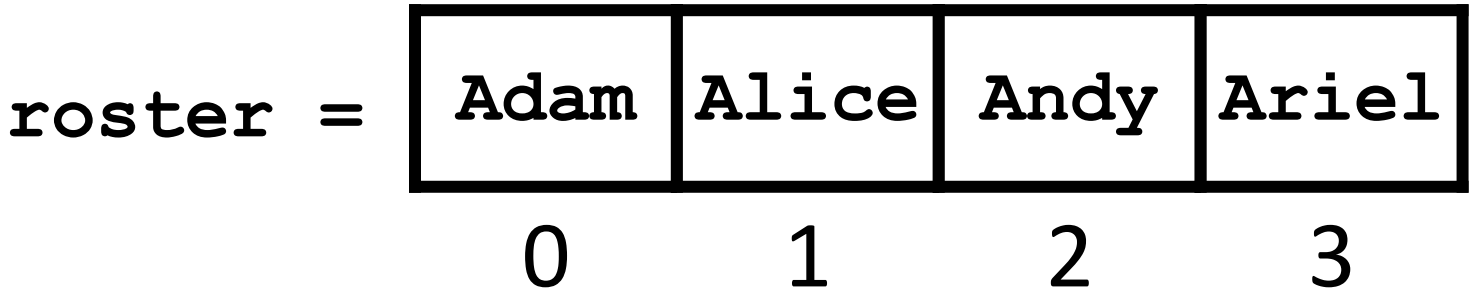
```
roster = ["Adam", "Alice", "Andy", "Ariel"]
```



Example of `remove()`

- We can use `remove()` to remove students who have dropped the class from the roster

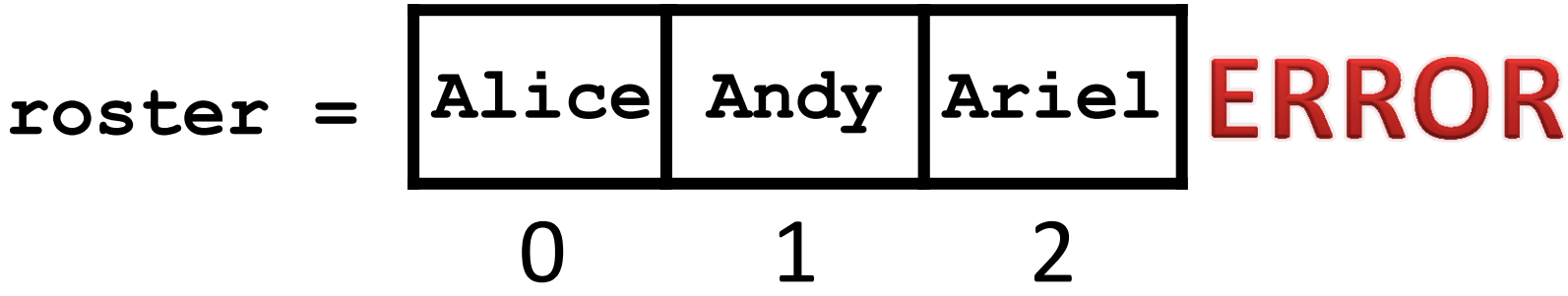
```
roster = ["Adam", "Alice", "Andy", "Ariel"]
roster.remove("Adam")    # Adam has dropped the class
```



Example of `remove()`

- We can use `remove()` to remove students who have dropped the class from the roster

```
roster = ["Adam", "Alice", "Andy", "Ariel"]
roster.remove("Adam")    # Adam has dropped the class
roster.remove("Bob")     # Bob is not in the class
```



Interactive `while` Loops

When to Use `while` Loops

- **`while`** loops are very helpful when you:
  - Want to get input from the user that meets certain specific conditions
    - Positive number
    - A non-empty string
  - Want to keep getting input until some "end"
    - User inputs a value that means they're finished
    - Reached the end of some input (a file, etc.)

Example `while` Loop

- We can use a **`while`** loop to get correct input from the user by re-prompting them

```
num  = 0          # we have to initialize num to zero

while num <= 0:          # so that we can use it here
    num = int(input("Enter a positive number: "))

# the while loop has exited b/c num is positive
print("Thank you.  The number you chose is:", num)
```

Nested Loops

Nesting

- You have already used nested statements
  - In HW3, you used nested **`if/elif/else`** statements to help you diagnose a patient

- We can also nest loops!
  - First loop is the *outer loop*
  - Second loop is the *inner loop*

Nested Loop Example

- What does this code do?

```
scores = []
for i in range(10):
    num = 0

    while num <= 0:
        num = int(input("Enter a positive #: "))
    scores.append(num)

print(scores)
```

Nested Loop Example

- What does this code do?

```
scores = []
for i in range(10):
    num = 0

    while num <= 0:
        num = int(input("Enter a positive #: "))
    scores.append(num)

print(scores)
```

creates an empty list

will run 10 times

will keep running until **num** is positive

after the **while** loop exits, **num** is positive, so add it to the **scores** list

Two-Dimensional Lists

Two-Dimensional Lists

- We've looked at lists as being one-dimensional
  - But lists can also be two- (or three- or four- or five-, etc.) dimensional!
- Lists can hold any type (int, string, float, etc.)
  - This includes holding another list

Two-Dimensional Lists: A Grid

- May help to think of 2D lists as a grid

```
twoD = [ [1,2,3], [4,5,6], [7,8,9] ]
```

1	2	3
4	5	6
7	8	9

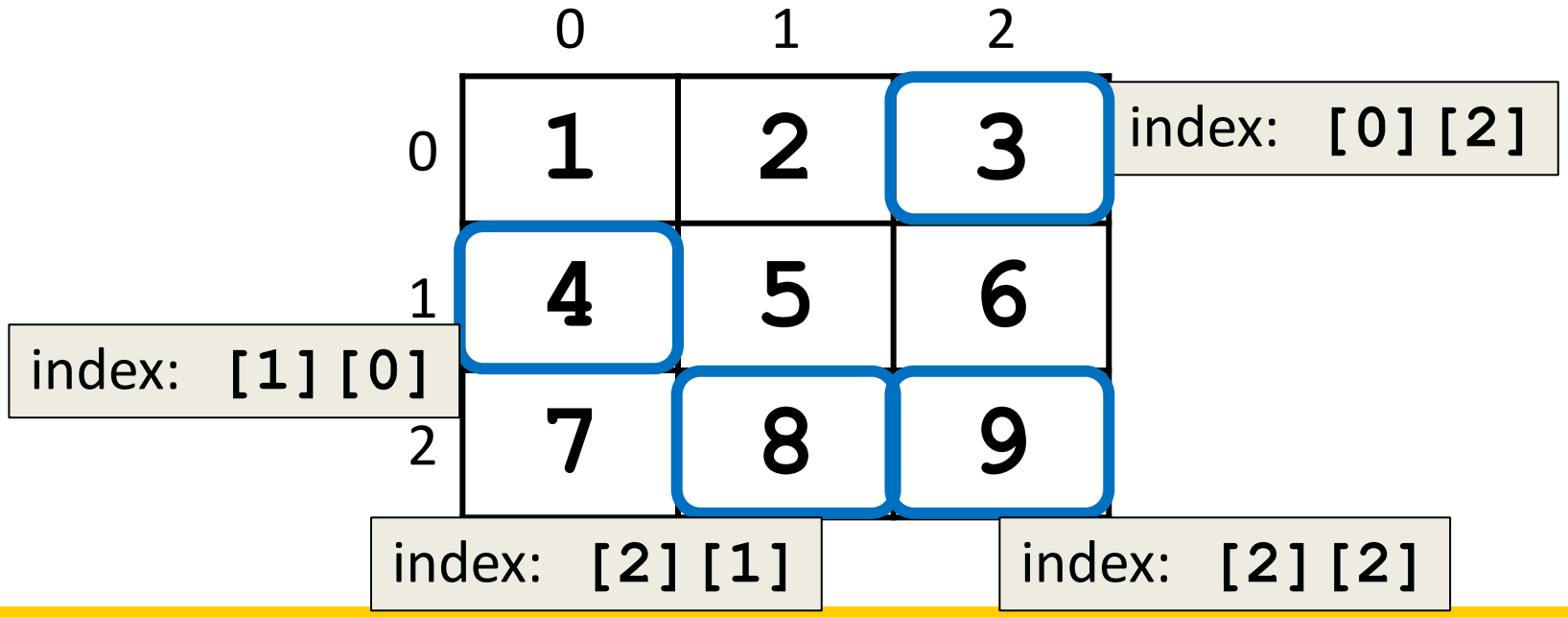
Two-Dimensional Lists: A Grid

- You access an element by the index of its row, then column (remember – index starts at 0!)

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Two-Dimensional Lists: A Grid

- You access an element by the index of its row, then column (remember – index starts at 0!)



Lists of Strings

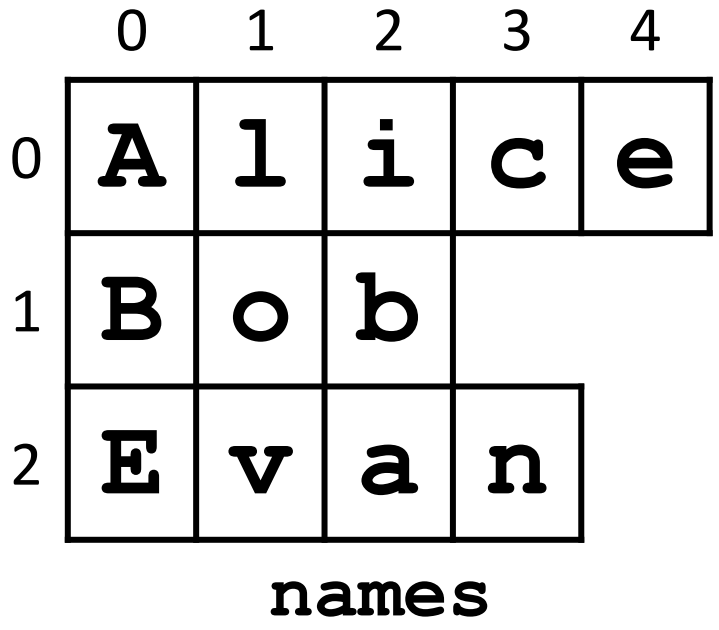
- Remember, a string is a list of characters
- So what is a list of strings?
  - A two-dimensional list!

- We have the index of the string (the row)
- And the index of the character (the column)

Lists of Strings

- Lists in Python don't have to be rectangular
  - They can also be jagged (rows different lengths)

- Anything we could do with a one-dimensional list, we can do with a two-dimensional list
  - Slicing, index, appending



NOTE: Strings vs Lists of Characters

- Strings and lists of characters do not behave the same way in Python; they have different functions, and different things that are allowed

- Strings – can use **`upper()`** and **`lower()`**

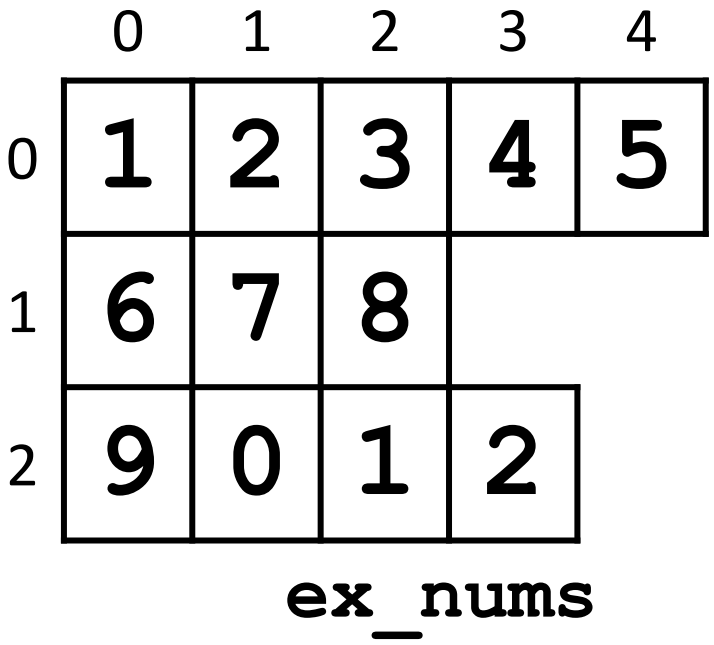
```
names = ['Alice', 'Bob', 'Evan']
```

- List of characters – can use **`append()`**

```
names = [list("Alice"), list("Bob"), list("Evan")]
[['A', 'l', 'i', 'c', 'e'], ['B', 'o', 'b'],
 ['E', 'v', 'a', 'n']]
```

Practice: Two-Dimensional Lists

1. Using a loop, print all five numbers from the first row of **`ex_nums`**
2. Replace the 4 with the word “four”
3. Add a 3 to the end of the last row
4. Delete the 5 from the list



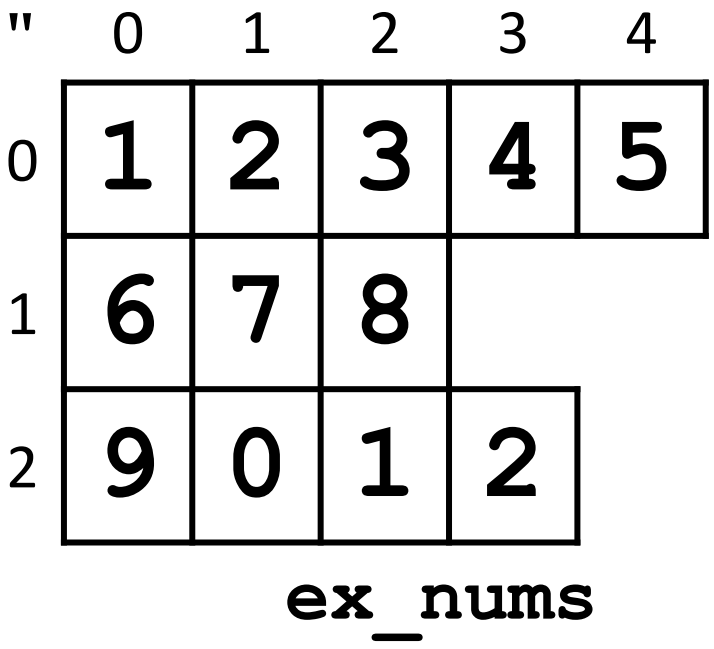
Answers: Two-Dimensional Lists

1.
```
for i in ex_nums[0]:
    print(i)
```

2. `ex_nums[0][3] = "four"`

3. `ex_nums[2].append(3)`

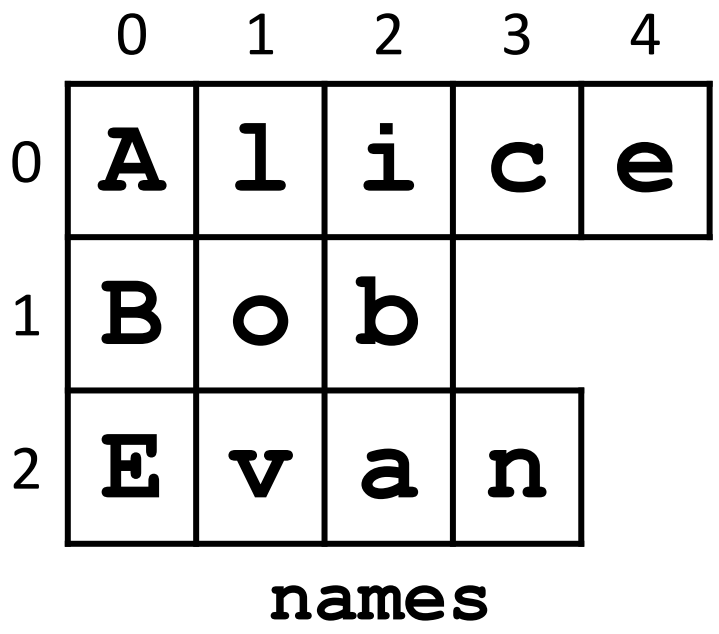
4. `ex_nums[0].remove(5)`



Practice: List of Lists of Characters

1. Add a “b” and a “y” to the end of “Bob”
2. Print out the second letter in “Evan”
3. Change “Alice” to “Alyce”

```
names[1].append('b')
names[1].append('y')

print(names[2][1])

names[0][2] = 'y'
```



Announcements

- (Pre) Lab 5 has been released on Blackboard
  - Future ones will be available the weekend prior
- Homework 4 is out
  - Due by Tuesday (Oct 6th) at 8:59:59 PM
- Homework 1 re-grade and re-submit petitions must be made to your TA before Friday @ 3 PM