

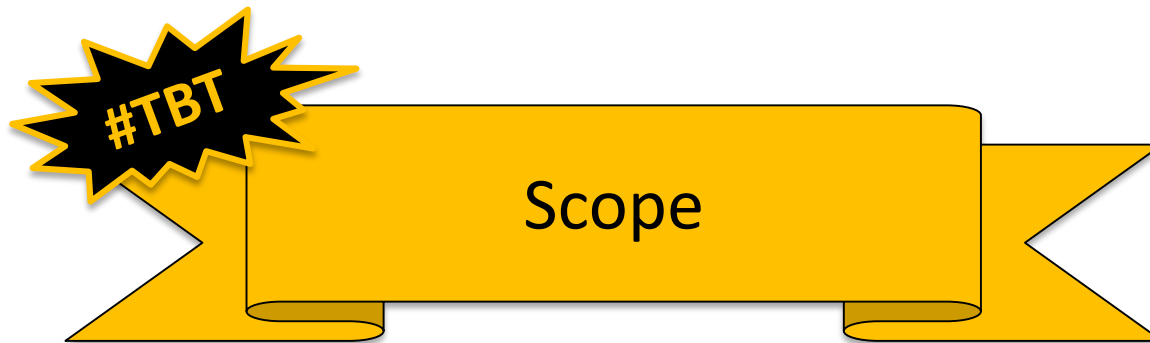
CMSC201

Computer Science I for Majors

Lecture 15 – For Loops

Last Class We Covered

- Two-dimensional lists
- Lists and functions
- Mutability



Any Questions from Last Time?

Today's Objectives

- To learn about and be able to use a **for** loop
 - To understand the syntax of a **for** loop
 - To use a **for** loop to iterate through a list
- To learn about the **range()** function
- To be able to combine **range()** and **for**
- To create a 2D list using loops



Control Structures (Review)

- A program can proceed:
 - In sequence
 - Selectively (branching): make a choice
 - Repetitively (iteratively): looping
 - By calling a function

focus of
today's lecture

Looping

- Python has two kinds of loops, and they are used for two different purposes
- The **while** loop
 - Works for basically everything
- The **for** loop:
 - Best at *iterating* over a list
 - Best at counted iterations

what we're covering today

for Loops: Iterating over a List

Iterating Through Lists

- **Iteration** is when we move through a list, one element at a time
 - Iteration is best completed with a loop
 - We did this previously with our **while** loop
- Using a **for** loop will make our code much faster and easier to write
 - Even faster than the **while** loop was to write!

Parts of a `for` Loop

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

```
myList = ['a', 'b', 'c', 'd']
```

```
for listItem in myList:  
    print(listItem)
```

Parts of a `for` Loop

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

initialize the list

```
myList = ['a', 'b', 'c', 'd']
```

how we will refer
to each element

the list we want
to iterate through

```
for listItem in myList:
```

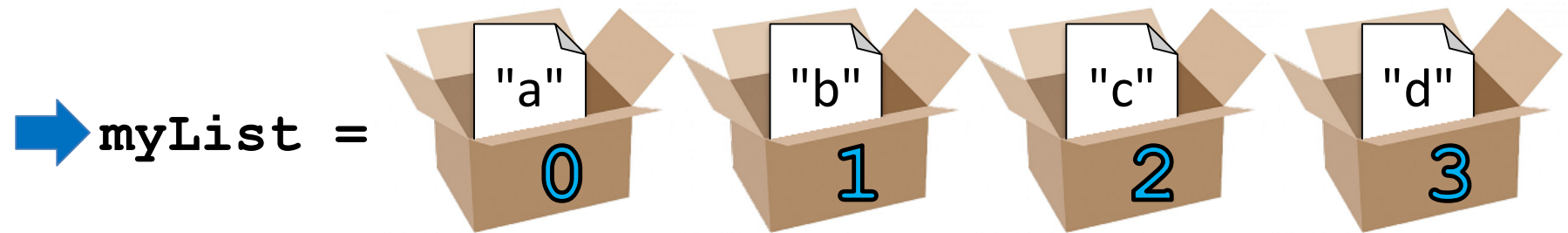
```
    print(listItem)
```

the *body* of the loop

How a `for` Loop Works

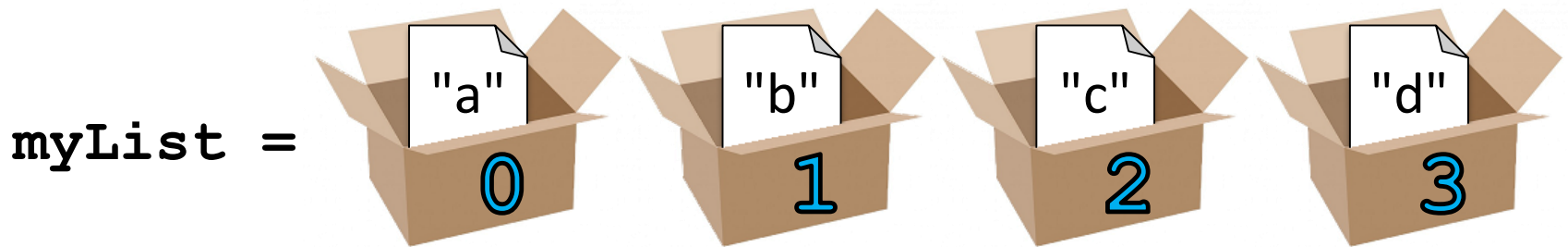
- In the `for` loop, we declared a new variable called `listItem`
 - The loop changes this variable for us
- The first time through the loop, `listItem` will be the value of the first element of the list (`'a'`)
- The second time through the loop, `listItem` will be the value of the second element of the list (`'b'`)
- And so on...

for Loop Explanation



```
for listItem in myList:  
    print(listItem)
```

for Loop Explanation



➔ `for listItem in myList:`
`print(listItem)`



for Loop Explanation

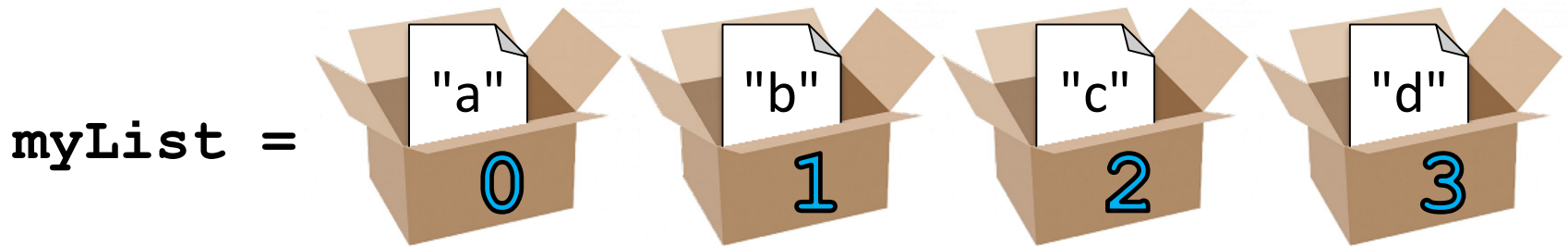


→ `for listItem in myList:`
`print(listItem)`

output: **a**



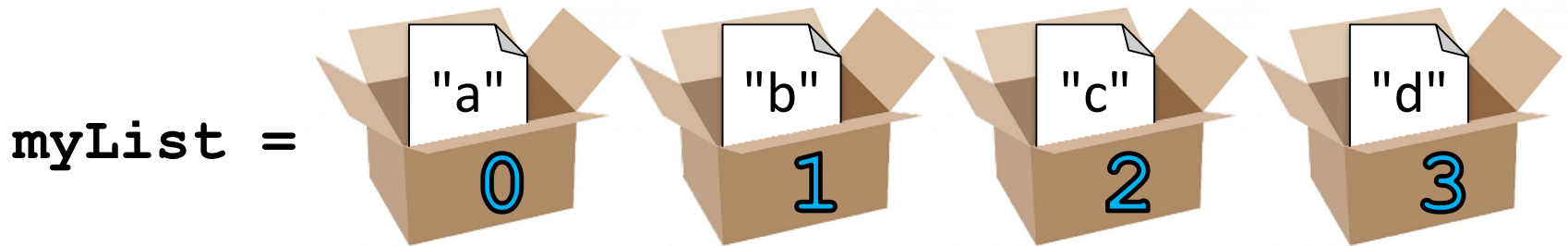
for Loop Explanation



➔ `for listItem in myList:
 print(listItem)`

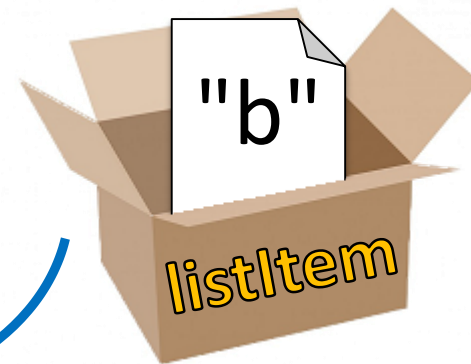


for Loop Explanation

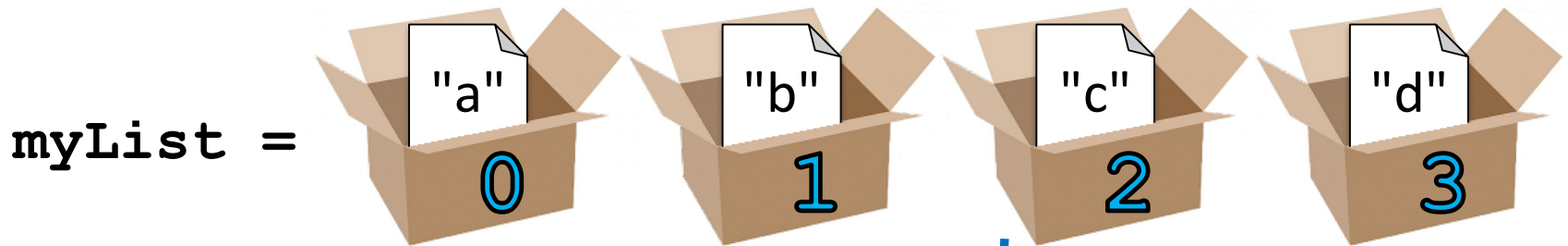


→ `for listItem in myList:`
`print(listItem)`

output: **b**



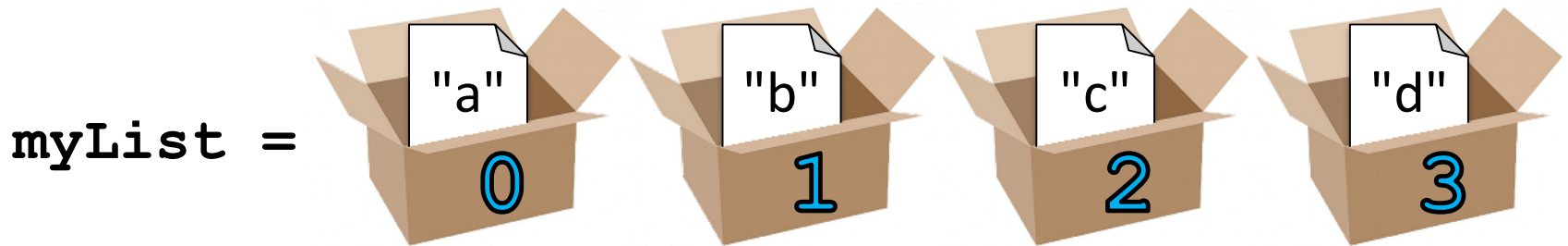
for Loop Explanation



➔ `for listItem in myList:`
`print(listItem)`

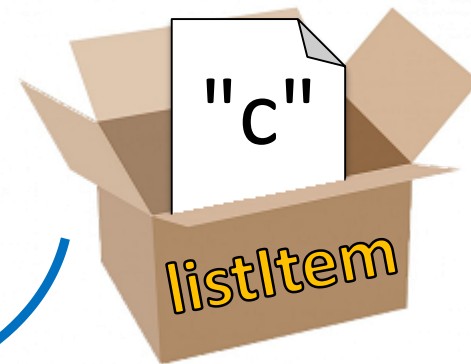


for Loop Explanation

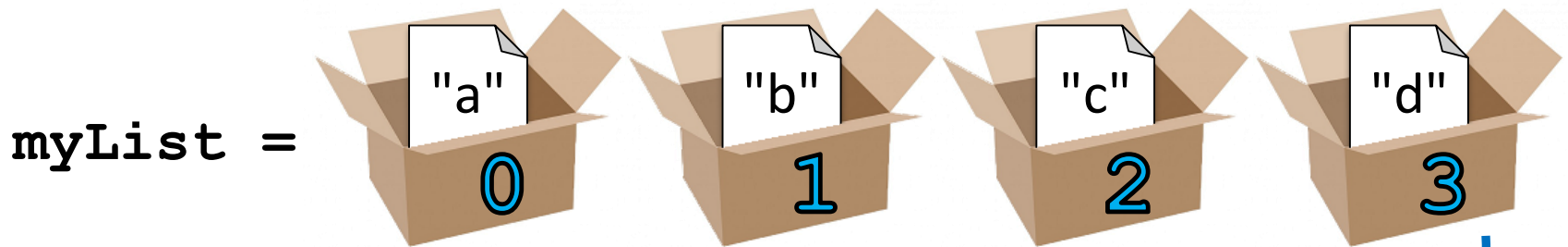


→ `for listItem in myList:`
`print(listItem)`

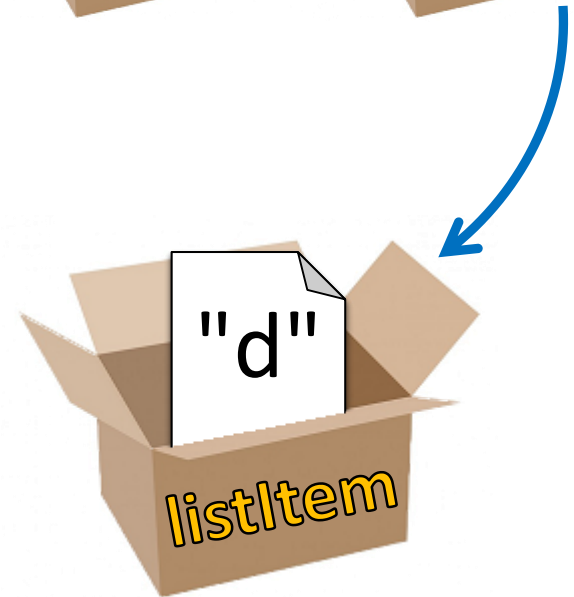
output: **c**



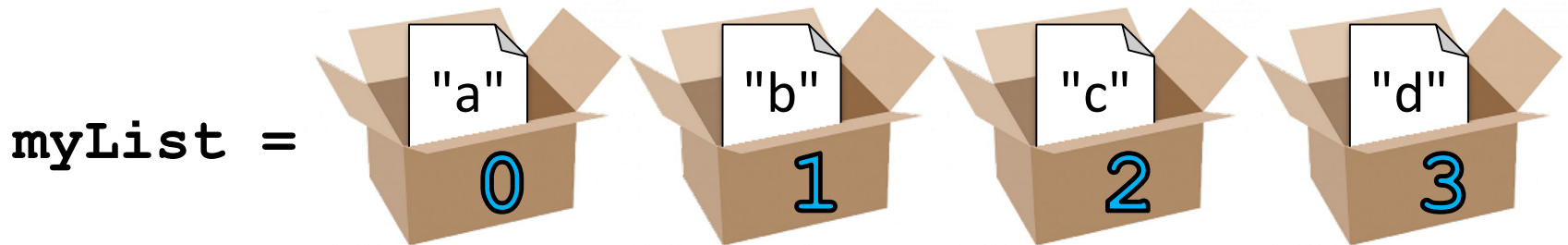
for Loop Explanation



➔ `for listItem in myList:`
`print(listItem)`

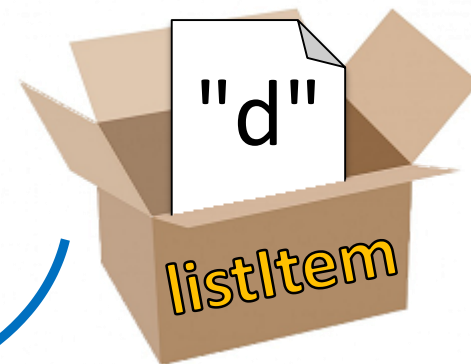


for Loop Explanation



→ `for listItem in myList:`
`print(listItem)`

output: **d**



Another Example `for` Loop

- Write code that uses 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)
```

Quick Note: Variable Names

- Remember, variable names should always be meaningful
 - And they should be more than one letter!
- There's one exception: loop variables

```
for n in nums:  
    sum = sum + n
```

 - The context for their name is clear
 - You can still make them longer if you want

Strings and `for` Loops

- We can use a `for` loop on strings as well

```
music = "jazz"
```

```
for c in music:  
    print(c)
```

```
j  
a  
z  
z
```

What will this code do?

- The `for` loop goes through the string letter by letter, and handles each one separately

The `for` Loop Variable

Updating Loop Variable

- What do you think this code does?

```
myList = [1, 2, 3, 4]
for listItem in myList:
    listItem = 4
print("List is now:", myList)
```

```
List is now: [1, 2, 3, 4]
```

“Copying” the List Elements

- The loop variable is a separate “box” from the elements of the list itself
 - It’s only a copy of each element’s value
- Editing `listItem` doesn’t change the actual contents of `myList`
 - There is a way to do this, though!

“Copying” the List Elements

- The `for` loop is essentially doing this:

```
listItem = myList[0]
```

```
listItem = 4
```

```
listItem = myList[1]
```

```
listItem = 4
```

```
# and so on...
```

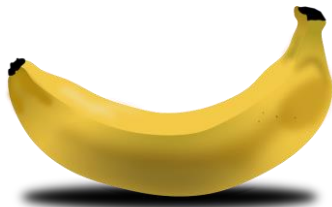
- You can see now why this doesn't change the list

Practice: Printing a List

- Given a list of strings called `food`, use a `for` loop to print out that each food is yummy!

```
food = ["apples", "bananas", "cherries", "durians"]  
# for loop goes here  
for f in food:  
    print(f, "are yummy!")
```

```
apples are yummy!  
bananas are yummy!  
cherries are yummy!  
durians are yummy!
```



The `range()` function

Range of Numbers

- Python has a built-in function called `range()` that can generate a list of numbers

cast it to a list to force it generate the numbers now

```
ex = list(range(0, 10))  
print(ex)
```

like slicing – it's UP TO
(but not including) 10

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Syntax of `range()`

`range(start, stop, step)`

the name of
the function

the number we
want to start
counting at

the number we want
to count UP TO (but
will not include)

how much we
want to count by

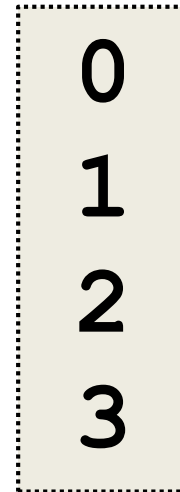
Examples of `range()`

- There are three ways we can use `range()`
- With one number
`range(10)`
- With two numbers
`range(10, 20)`
- With three numbers
`range(10, 20, 2)`

`range ()` with One Number

- If `range ()` is given only one number
 - It will start counting at 0
 - And will count up to (but not including) that number
 - Incrementing by one

```
for p in range (4) :  
    print (p)
```



0
1
2
3

range () with Two Numbers

- If we give it two numbers, it will count from the first number up to the second number

```
for a in range(5,10):
```

```
    print(a)
```



5

6

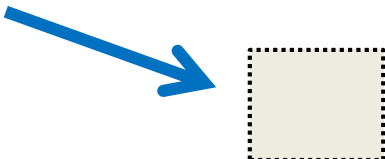
7

8

9

```
for b in range(10,5):
```

```
    print(b)
```



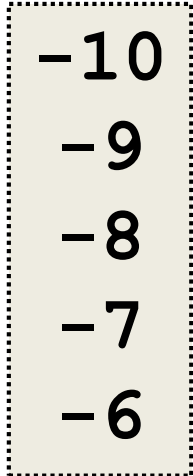
range () counts
up by default!

range () with Two Numbers

- If we give it two numbers, it will count from the first number up to the second number

```
for c in range(-10, -5):  
    print(c)
```

from a lower to a
higher number



-10
-9
-8
-7
-6

`range ()` with Three Numbers

- If we give it three numbers, it will count from the first number up to the second number, and it will do so in steps of the third number

```
>>> threeA = list(range(2, 11, 2))
>>> print(threeA)
[2, 4, 6, 8, 10]
>>> threeB = list(range(3, 28, 5))
>>> print(threeB)
[3, 8, 13, 18, 23]
```



`range ()` starts counting at the first number!

Counting Down with `range()`

- By default, `range()` counts up
 - But we can change this behavior
- If the **STEP** is set to a negative number, then `range()` can be used to count down

```
>>> downA = list(range(10, 0, -1))
```

```
>>> print(downA)
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Using `range()` in `for` Loops

- We can use the `range()` function to control a loop through “counting”

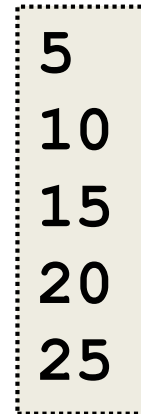
```
for i in range(0, 20):  
    print(i + 1)
```

- What will this code do?
 - Print the numbers 1 through 20 on separate lines
- The `for` loop is still iterating over a list


Using `range()` in `for` Loops

- When we use the `range()` function in `for` loops, we don't need to cast it to a list
 - The `for` loop handles that for us

```
print("Counting by fives...")
for num in range(5, 26, 5):
    print(num)
```



5
10
15
20
25



call the `range()` function, but
don't need to cast it to a list

Combining `for` and `range()`

Using a `for` Loop with `range()`

- We can combine a simple `for` loop with the `range()` function, as shown below

```
for i in range( len(theList) ):
    print( theList[i] )
```

- What's the benefit to doing it this way?
- Why do we need `range()` and `len()`?
 - We'll answer these questions momentarily

Contents vs Indexes

- Previously, we had used the `for` loop to iterate over the contents of the list
 - For example: “a”, “b”, “c”, “d”
- Just now, we used the `for` loop to iterate over the indexes of the list
 - For example: 0, 1, 2, 3
- Both examples are iterating over a list

Why `range()` and `len()`?

- Why do we need `len()`?
 - To know how many indexes the list has
 - It will give us an integer value
- Why do we need `range()`?
 - To generate all the indexes of the list
- What does `range()` do with one number?
 - Start at 0, and count up to the number given

Common Error

- Pay attention with `len()` and `range()`
- Which goes on the outside?
 - `range()`
 - It needs the length to generate the indexes
- If you use them backwards:
`TypeError: 'list' object cannot be interpreted as an integer`

Time for...

LIVECODING!!!

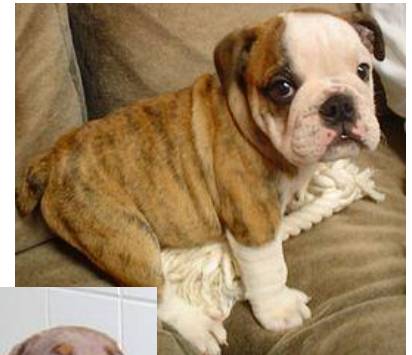
Running a Kennel

- You are running a kennel with space for 5 dogs
- You ask your 3 assistants to do the following, using the list of dogs in your office:
 1. Tell you all of the dogs in the kennel
 2. Tell you what pen number each dog is in
 3. Later, all the dogs have been picked up, and someone dropped off their 5 German Shepherds, so the list in your office needs to be updated

Running a Kennel

- The dogs in your kennel at the start are:

Alaskan Klee Kai	Beagle	Chow Chow	Doberman	English Bulldog
---------------------	--------	--------------	----------	--------------------



Using Loops to Make 2D Lists

- The easiest way to create a 2D list is to
 - Start with an empty one-dimensional list
 - Create the first “row” as a separate list
 - Append it to the original 1D list
 - Repeat until all rows are added to the list
- You can use a **while** loop, but **for** loops are great at creating lists of a specific size

Example: Creating 2D List

- Create a 6-high by 4-wide list of underscores

```
board = []  
row   = ["_", "_", "_", "_"]
```

```
for i in range(6):  
    board.append( row[:] )
```

why is this here?

each row needs to be individual, hence it needs to be deep copied

Example: Creating 2D List from Input

- Create a list of names and majors for 5 students

```
info = []
```


```
for i in range(5):
```

```
    name = input("Enter name: ")
```

```
    major = input("Major? ")
```

```
    row = [name, major]
```

```
    info.append(row)
```



why doesn't this row need to be deep copied?

Announcements

- HW 5 out on Blackboard
 - Must re-take the Academic Integrity Quiz to see it
 - Due Friday, April 7th @ 8:59:59 PM
- Discussions started again this week
 - Remainder of labs will be in-person
 - Pre Lab quizzes will come out Friday morning
- Final exam is Friday, May 19th from 6 to 8 PM