CMSC201
Computer Science I for Majors

Lecture 09 – For Loops

Last Class We Covered

- Lists and what they are used for
  - Operations a list can perform
  - Including **`append()`** and **`remove()`**
- Two dimensional lists
- Sentinel loops
  - Priming read (and non-priming read)

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 perform an action a set number of times
- To learn about the **`range()`** function
- To be able to combine **`range()`** and **`for`**

Looping (`for` Loops)

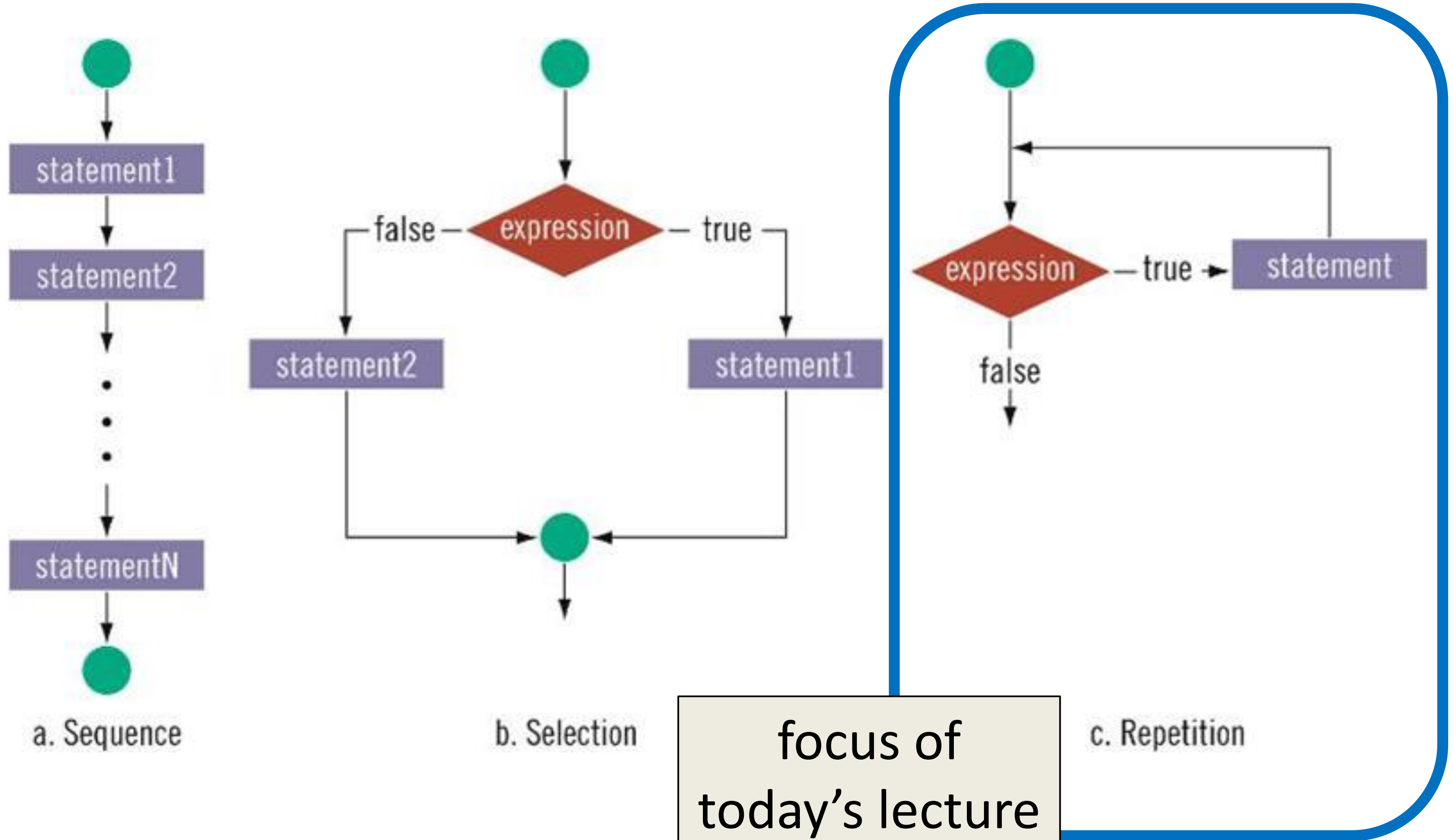
Image from wikipedia.org

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

Control Structures: Flowcharts



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

String Operators in Python

Operator	Meaning
`+`	Concatenation
`*`	Repetition
`STRING[#]`	Indexing
`STRING[#:#]`	Slicing
`len(STRING)`	Length
`for VAR in STRING`	Iteration

covering today

Review: Lists and Indexing

Review: List Syntax

- Use `[]` to assign initial values (*initialization*)

```
myList = [1, 3, 5]
words  = ["Hello", "to", "you"]
```

- And to refer to individual elements of a list

```
>>> print(words[0])
Hello
>>> myList[0] = 2
```

Review: Indexing in a List

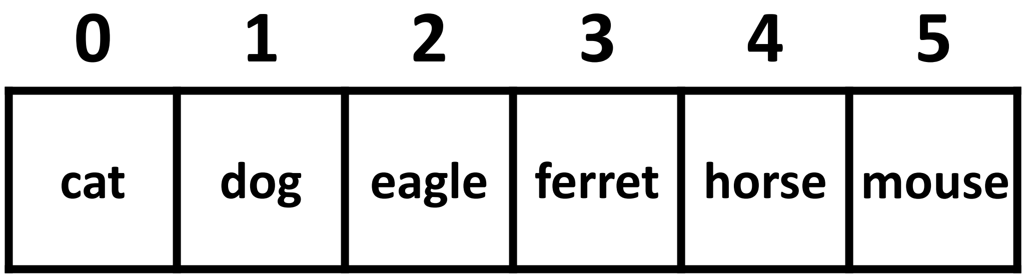
- Remember that list indexing starts at **zero**, not 1!

0	1	2	3	4	5
???	???	???	???	???	???

```
animals = ['cat', 'dog', 'eagle', 'ferret',
           'horse', 'lizard']
print("The best animal is a", animals[3])
animals[5] = "mouse"
print("The little animal is a", animals[5])
print("Can a", animals[4], "soar in the sky?")
```

Review: Indexing in a List

```
The best animal is a ferret
The little animal is a mouse
Can a horse soar in the sky?
```



```
animals = ['cat', 'dog', 'eagle', 'ferret',
           'horse', 'lizard']
print("The best animal is a", animals[3])
animals[5] = "mouse"
print("The little animal is a", animals[5])
print("Can a", animals[4], "soar in the sky?")
```

`for` Loops: Iterating over a List

Remember our Grocery List?

```
def main():
    print("Welcome to the Grocery Manager 1.0")
    # initialize the value and the size of our list
    grocery_list = [None]*3

    grocery_list[0] = input("Please enter your first item:  ")
    grocery_list[1] = input("Please enter your second item: ")
    grocery_list[2] = input("Please enter your third item:  ")
    print(grocery_list[0])
    print(grocery_list[1])
    print(grocery_list[2])
main()
```

We used a **`while`** loop to fix this, but we can also use a **`for`** loop

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

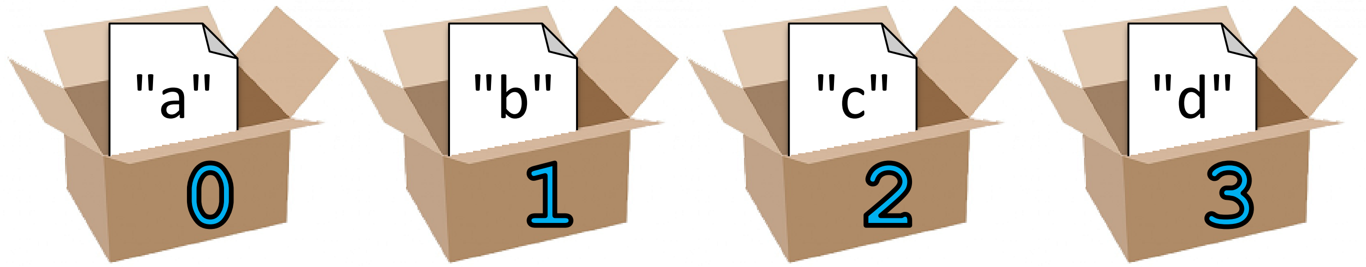
How a `for` Loop Works

- 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 first element of the list (`'a'`)
- The second time through the loop, `listItem` will be the second element of the list (`'b'`)
- And so on…

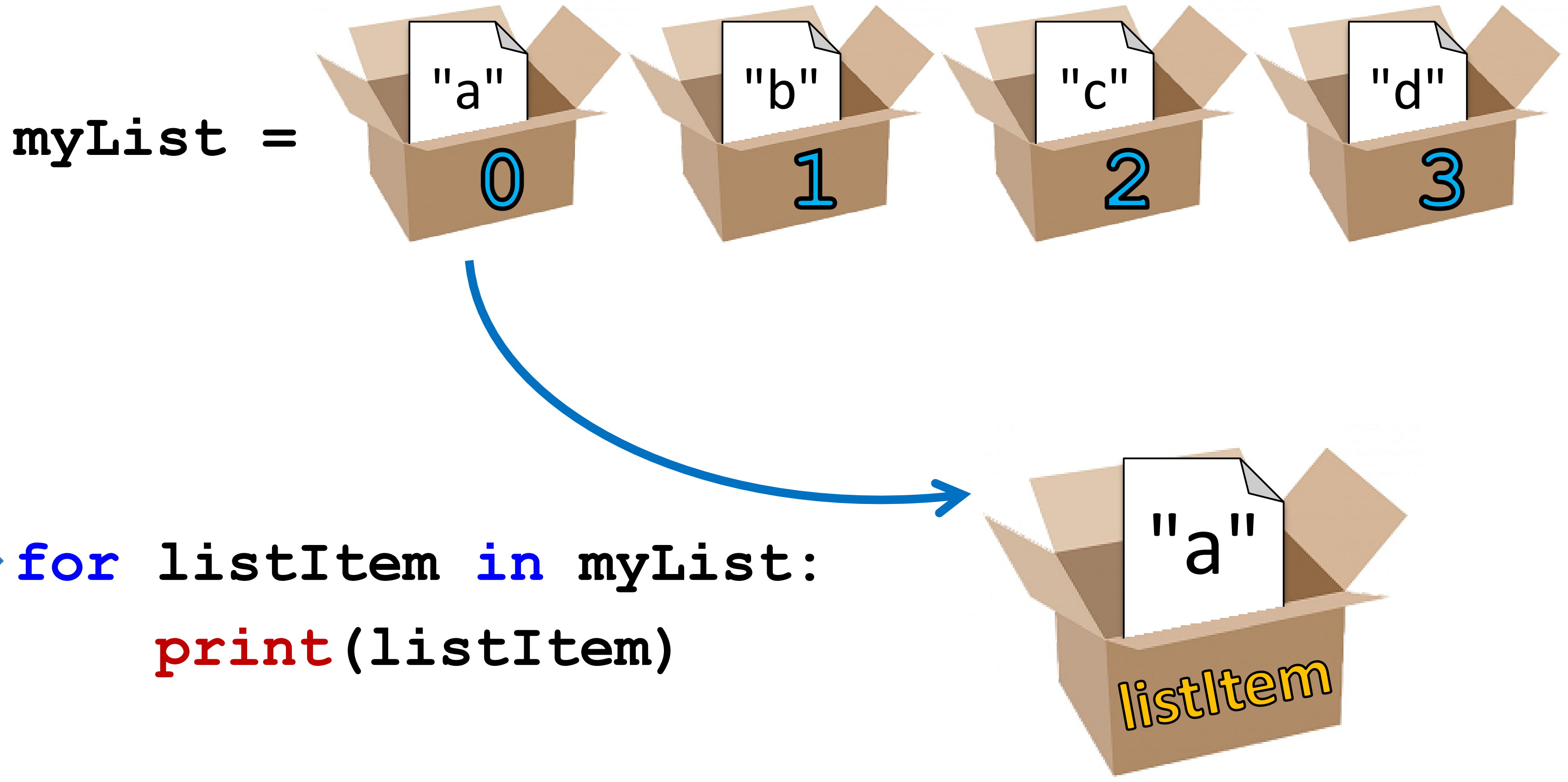
for Loop Explanation

myList =



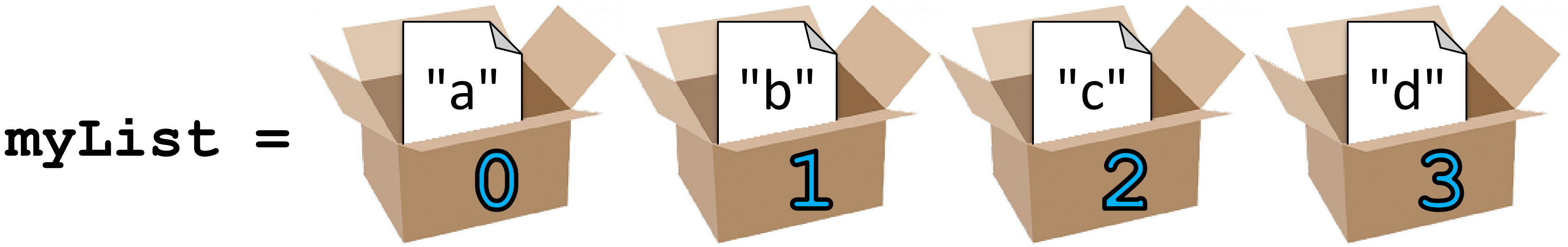
```
for listItem in myList:
    print(listItem)
```

for Loop Explanation

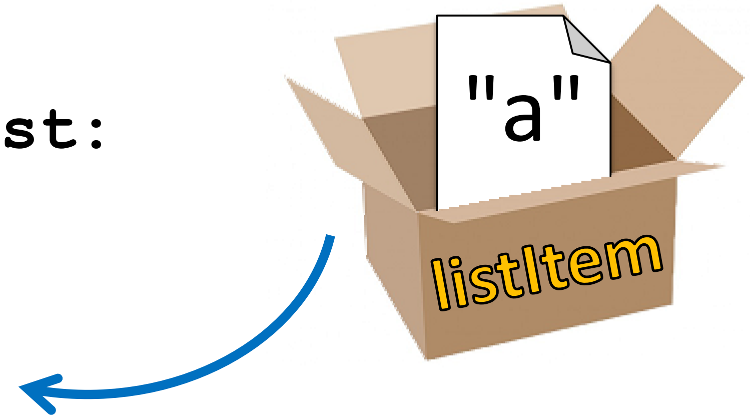


```
for listItem in myList:
    print(listItem)
```

`for` Loop Explanation

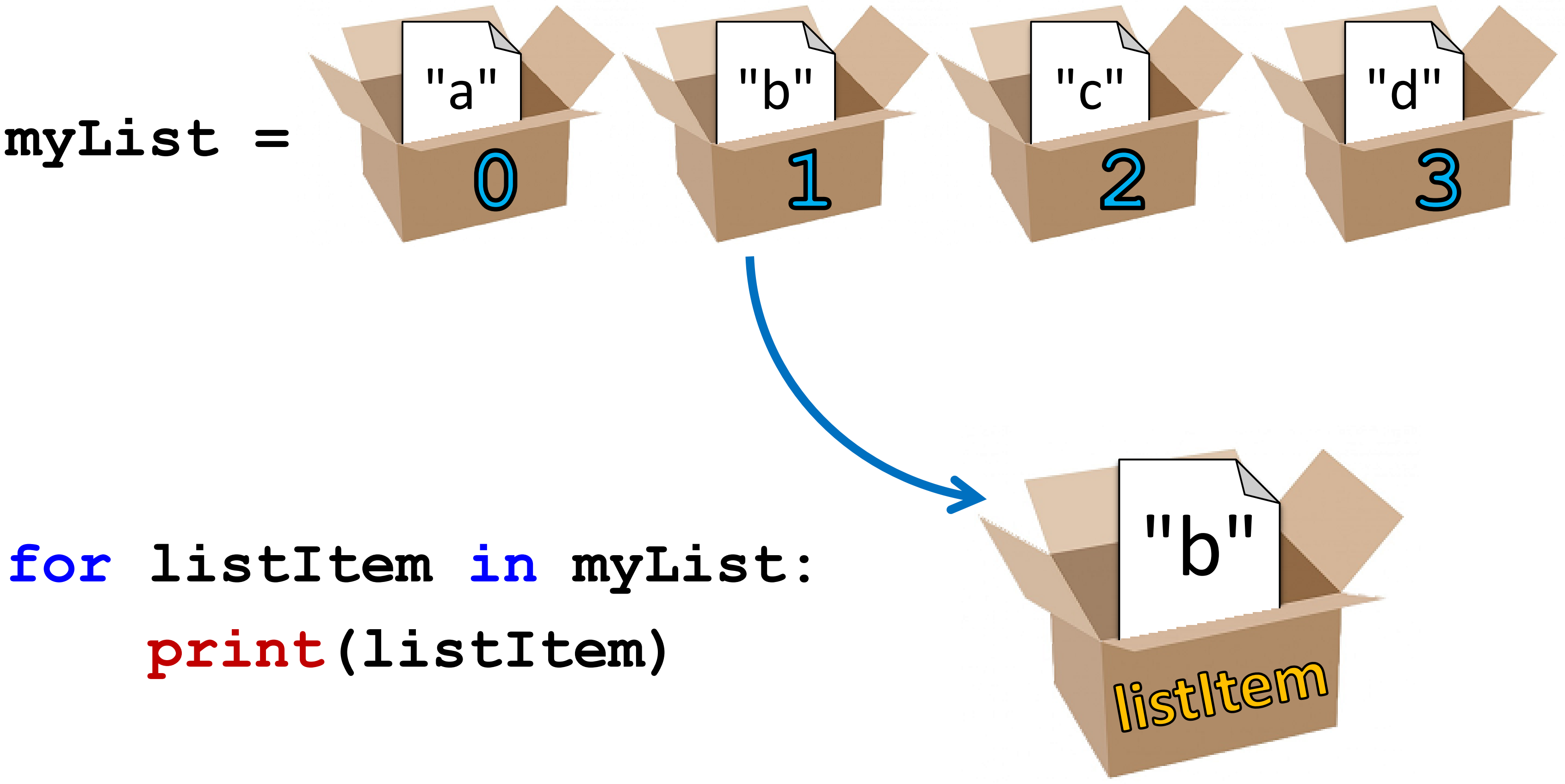


```
for listItem in myList:
    print(listItem)
```



```
output: a
```

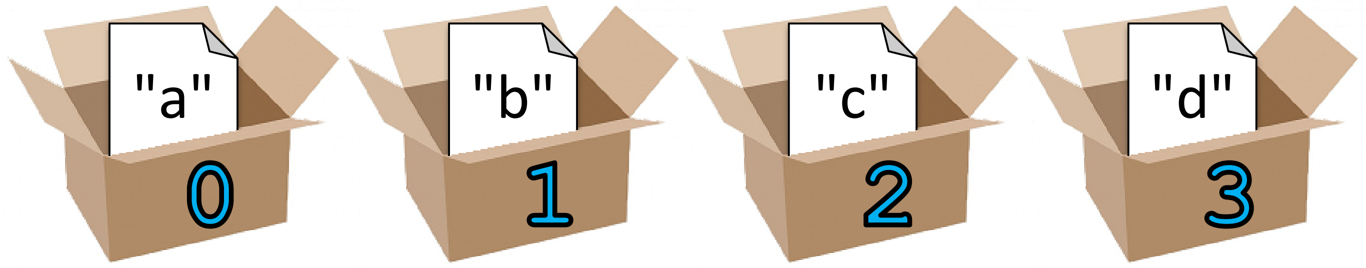
for Loop Explanation



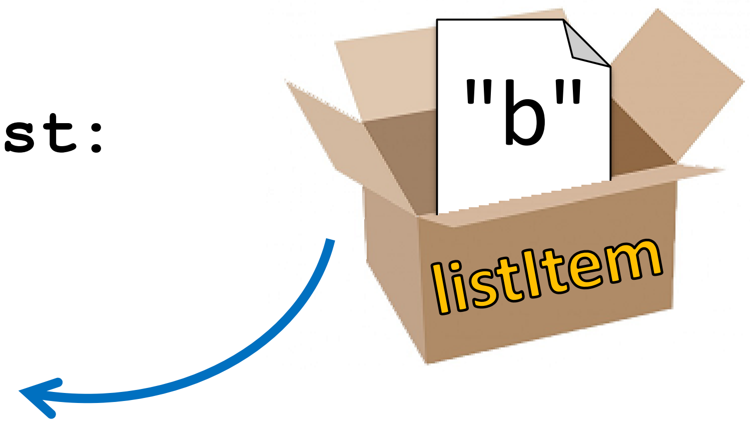
```
for listItem in myList:
    print(listItem)
```

for Loop Explanation

myList =

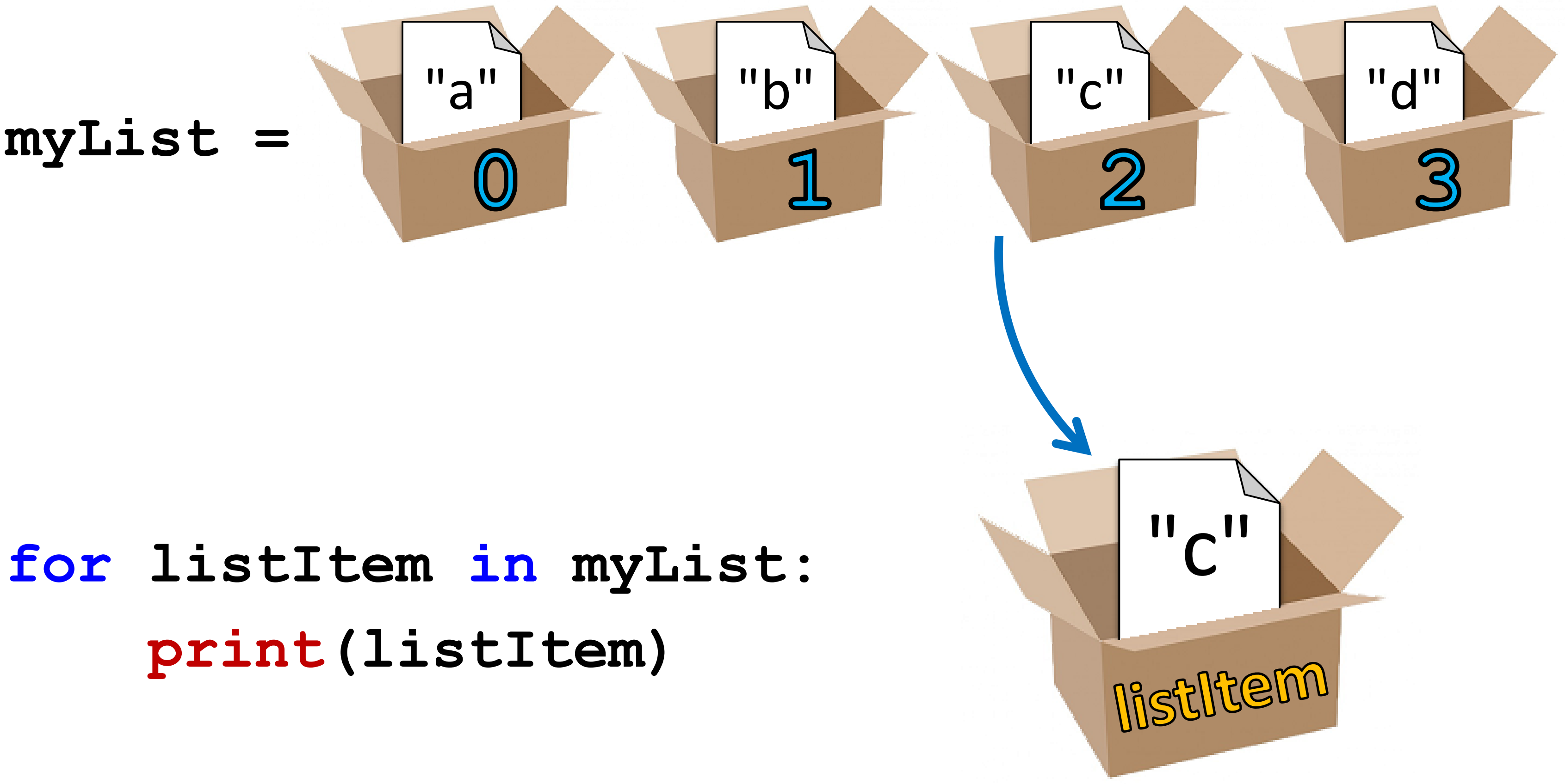


```
for listItem in myList:
    print(listItem)
```



```
output: b
```

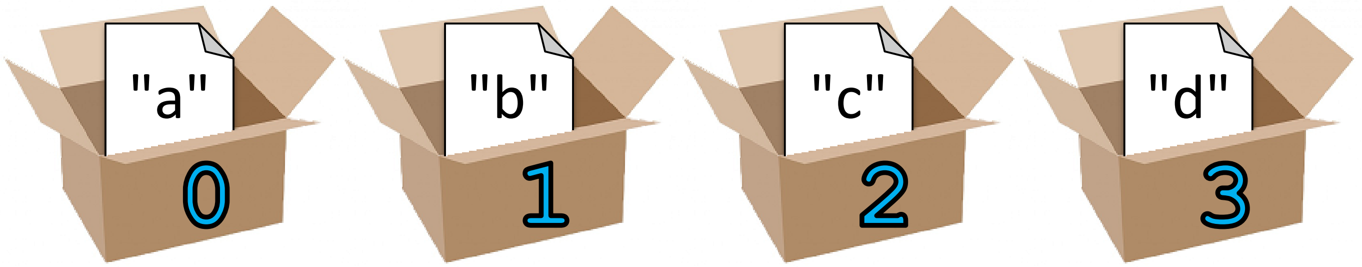
for Loop Explanation



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

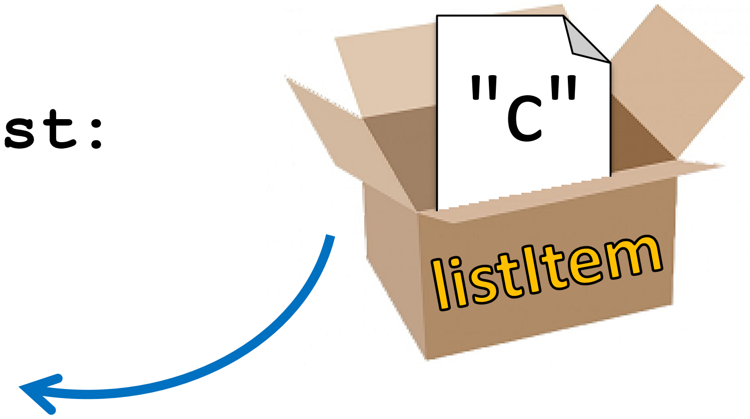
`for` Loop Explanation

`myList =`



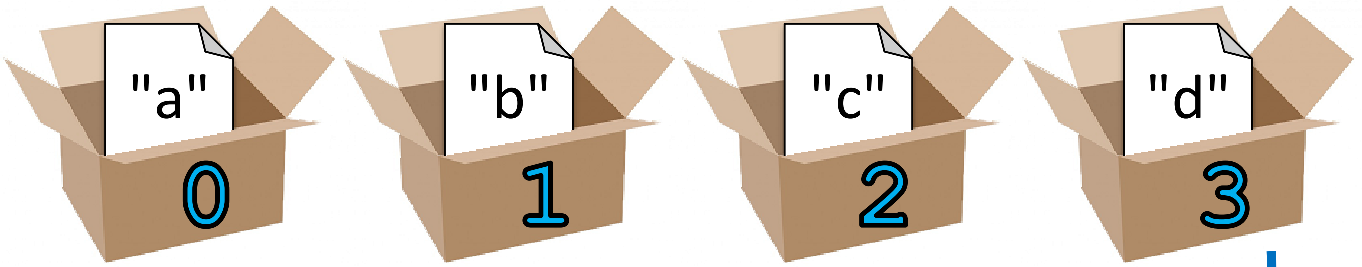
```
for listItem in myList:
    print(listItem)
```

output: c

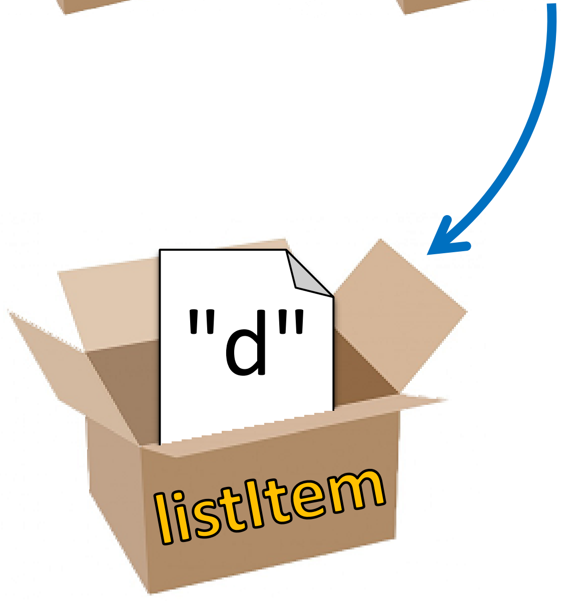


for Loop Explanation

```
myList =
```

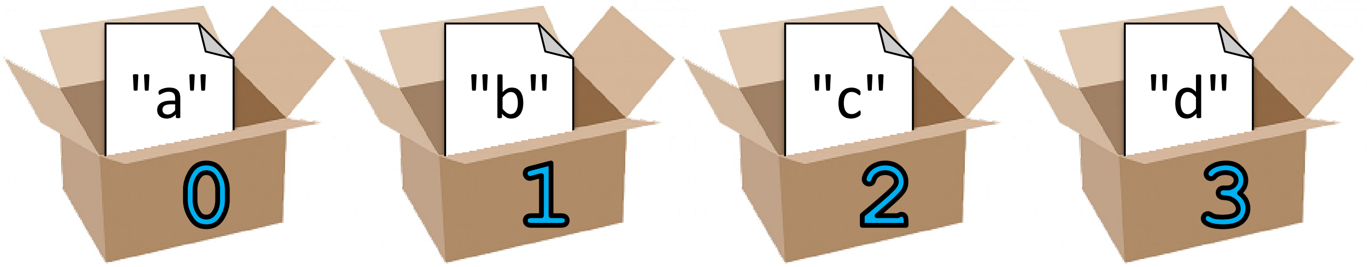


```
for listItem in myList:
    print(listItem)
```



for Loop Explanation

myList =



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

output: d

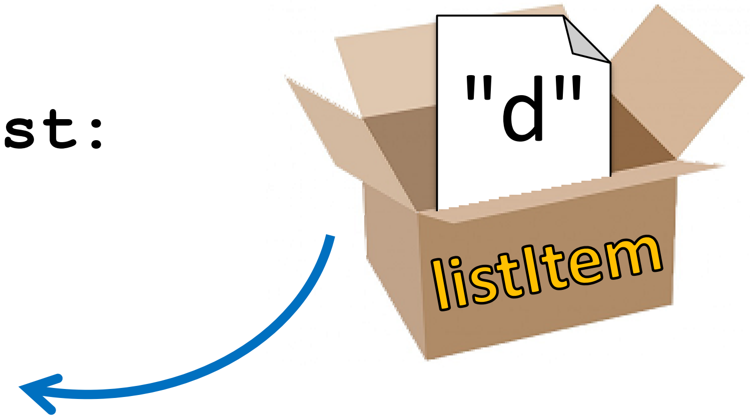


Image from pixabay.com

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 <u>**always**</u> be meaningful
  - And they should be more than one letter!
- There's one exception: loop variables

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

  - You can still make them longer if you want

Strings and `for` Loops

- Strings are represented as lists of characters
  - So we can use a **`for`** loop on them 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

- 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 from before is sort of doing this:

```
listItem = myList[0]
listItem = 4
listItem = myList[1]
listItem = 4
# and so on...
```

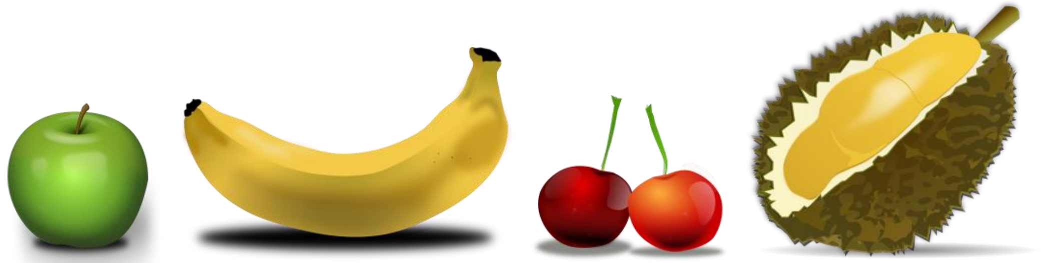
- You can see now why this doesn’t change the list

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 <u>not</u> include)

how much we want to count by

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

- 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

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(0, 100, 5)
```

`range()` with One Number

- If `range()` is given only one number
  - It will start counting at 0
  - And will count **<u>up to</u>** (but not including) that number
  - Counting by ones

```
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)
```

```
-10
-9
-8
-7
-6
```

from a lower to a higher number

`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 <u>negative</u> 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

- 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)
```

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

```
5
10
15
20
25
```

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 **<u>contents</u>** of the list
  - For example: “a”, “b”, “c”, “d”

- Just now, we used it the `for` loop to iterate over the **<u>indexes</u>** of the list
  - For example: 0, 1, 2, 3

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

Images from wikimedia.org

Benefits of Iterating with `for`

- Iterating over the indexes allows us to actually access (and <u>change</u>!) the contents of the list

- The `for` loop handles the necessary updating of the loop variable for us
  - Python may be "stupid," but it won't make coding mistakes like we might

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 <u>up to</u> the number given

Common Error

- Pay attention with **`len()`** and **`range()`**
- Which goes on the outside?
  - **`range()`**
  - It needs the <u>length</u> to generate the indexes
- If you use them backwards:

```
TypeError: 'list' object cannot be
interpreted as an integer
```

Announcements

- Slides with problems are at the end of this lecture; use them to practice `for` loops
- Homework 4 is due Wednesday
  - Homework 2 grades will be sent to you soon
- The midterm exam is fast approaching…
  - October 19th and 20th!
  - We'll review in class on the 17th and 18th

Practice: The `range()` function

- What lists will the following code generate?
  1. `prac1 = list(range(50))`
  2. `prac2 = list(range(-5, 5))`
  3. `prac3 = list(range(1, 12, 2))`

Practice: The `range()` function

- What lists will the following code generate?

1. `prac1 = list(range(50))`

```
[0, 1, 2, 3, 4, 5, ..., 47, 48, 49]
```

a list from 0 to 49, counting by 1

2. `prac2 = list(range(-5, 5))`

```
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
```

3. `prac3 = list(range(1, 12, 2))`

```
[1, 3, 5, 7, 9, 11]
```

Practice: Odd or Even?

- Write code that will print out, for the numbers 1 through 20, whether that number is even or odd

Sample output:

```
The number 1 is odd
The number 2 is even
The number 3 is odd
```

Practice: Odd or Even?

- Write code that will print out, for the numbers 1 through 20, whether that number is even or odd

```
for num in range(1,21):
    # if odd, will be 1 (which is True)
    if (num % 2):
        print(num, "is odd")
    # if even, divides cleanly into 2
    else:
        print(num, "is even")
```

Practice: Indexing in a List

- Using the list **`names`**, code the following:

Alice	Bob	Eve

1. Print "Bob sends a message to Alice"
2. Change the first element of the list to Ann
3. Print "BobBobAnnEve"

Practice : Indexing in a List

- Using the list **`names`**, code the following:

0	1	2
Ann	Bob	Eve

1. Print “Bob sends a message to Alice”
2. Change the first element of the list to Ann
3. Print “BobBobAnnEve”

```
print(names[1], "sends a message to", names[0])
names[0] = "Ann"
print(names[1] + names[1] + names[0] + names[2])
# or      print(names[1]*2 + names[0] + names[2])
```

Practice Problem

- Write a program that asks the user for input:
  - The height and width of a parallelogram
- Print a parallelogram with those dimensions to the user’s screen

```
bash-4.1$ python lec8_practice1.py
What is the height of your parallelogram? 4
What is the length of your parallelogram? 7
*******
 *******
  *******
   *******
```

Practice Problem

- Write a program that asks the user to enter a positive integer, and calculates the sum of $1^2 + 2^2 + \ldots + n^2$ and prints it to the screen
  - For example, if the user entered 5, the program would calculate $1^2 + 2^2 + 3^2 + 4^2 + 5^2$ $= 1 + 4 + 9 + 16 + 25 =$ **55**

```
bash-4.1$ python lec8_practice2.py
Enter a positive integer: 4
The sum of the first 4 squares is 30.
```