

CMSC201

Computer Science I for Majors

Lecture 08 – For Loops

Prof. Jeremy Dixon

Last Class We Covered

- The potential security concerns of `eval()`
- Lists and what they are used for
- How strings are represented
- How to use strings:
 - Indexing
 - Slicing
 - Concatenate and Repetition

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
 - Or to perform an action a set number of times
- To learn about the **range()** function
- To update our grocery program from last time!

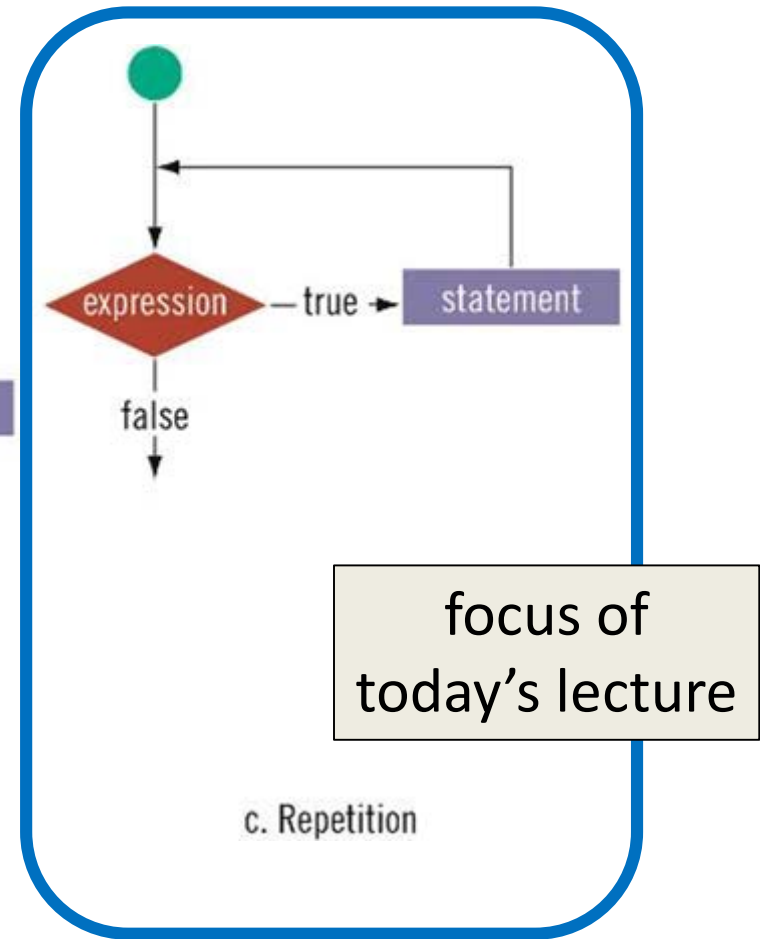
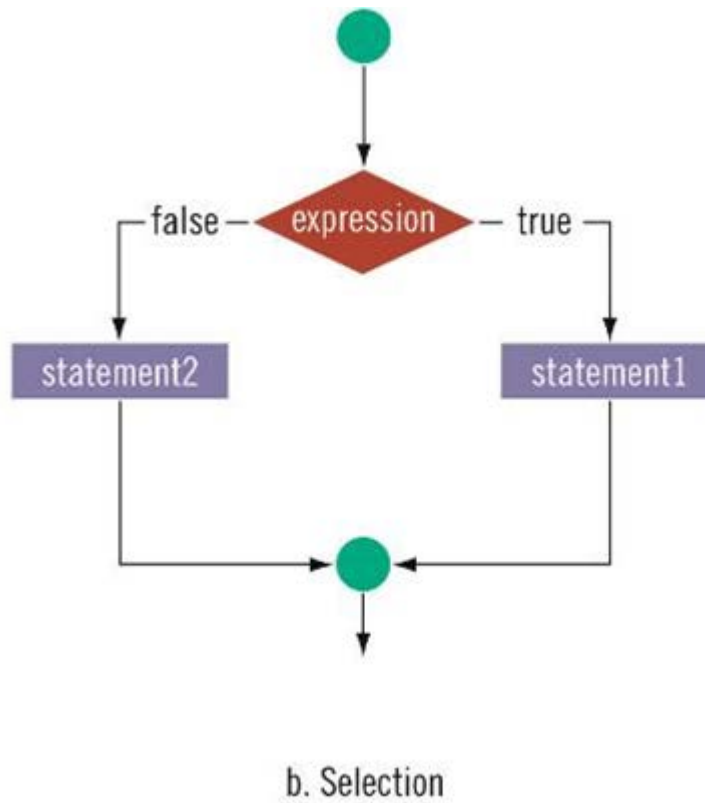
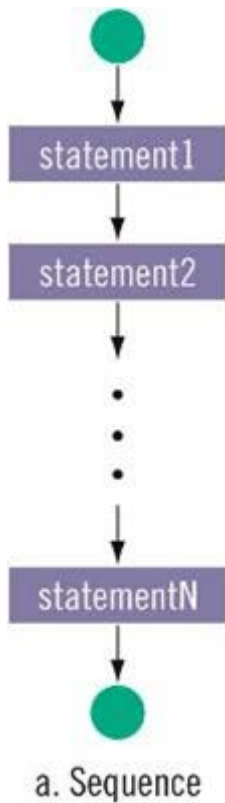
Looping

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



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

what we're
covering today

- The **while** loop

- Good for almost everything else

String Operators in Python

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

from last time

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

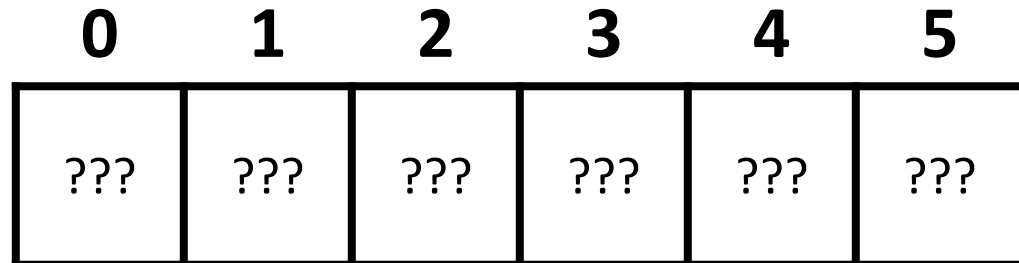
```
>>> print(words[0])
```

```
Hello
```

```
>>> myList[0] = 2
```

Review: Indexing in a List

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



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

Review: Indexing in a List

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

0	1	2	3	4	5
cat	dog	eagle	ferret	horse	mouse

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

Exercise: 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”

Exercise: Indexing in a List

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

0	1	2
Ann	Bob	Eve

- Print "Bob sends a message to Alice"
- Change the first element of the list to Ann
- 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])
```

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()
```

and that loops would
make this part easier?

Iterating Through Lists

- *Iteration* is when we move through a list, one element at a time
 - Instead of specifying each element separately, like we did for our grocery list
- Using a `for` loop will make our code much faster and easier 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 are declaring a new variable called “`listItem`”
 - The loop will change this variable for us
- The first time through the loop, `listItem` will be the first element of the list
- The second time through the loop, `listItem` will be the second element of the list
- And so on...

Example `for` Loop

- We can 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)
```

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
```

- You can (of course) make them longer if you want.

A Downside!

- 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]
```

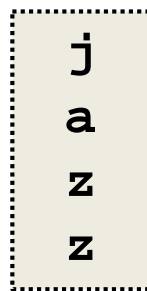
- Changing `listItem` does not change the original list!
 - It's only a copy of each element

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

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 the generator to run

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

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

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

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

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(0, 100, 5)`

`range()` with One Number

- If we just give it one number, it will start counting at 0, and will count UP TO that number

```
>>> one = list(range(4))
```

```
>>> one
```

```
[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

```
>>> twoA = list(range(5,10))
```

```
>>> twoA
```

```
[5, 6, 7, 8, 9]
```

```
>>> twoB = list(range(-10,-5))
```

```
>>> twoB
```


```
[-10, -9, -8, -7, -6]
```

```
>>> twoC = list(range(-5,-10))
```


```
>>> twoC
```

```
[]
```

from a lower to a higher number



range() can only count up!



`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))
```

```
>>> threeA
```

```
[2, 4, 6, 8, 10]
```

```
>>> threeB = list(range(3, 28, 5))
```

```
>>> threeB
```

```
[3, 8, 13, 18, 23]
```



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

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]
```

Counting Down with `range()`

- We said `range()` could only count up
 - But that's not strictly true!
- If the **STEP** is set to a negative number, then `range()` can be used to count down

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

```
>>> downA
```

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

```
>>> downB = list(range(18,5,-2))
```

```
>>> downB
```

```
[18, 16, 14, 12, 10, 8, 6]
```

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 (num % 2): # will be 1 (True)  
        print(num, + "is odd")  
    else: # divides cleanly into 2  
        print(num, + "is even")
```

Practice: Update our Grocery List!

- Remember from last time...
 - What would make this process easier?
 - Loops!
 - Instead of asking for each item individually, we could keep adding items to the list until we wanted to stop (or the list was “full”)
- Let’s do this!

LIVECODING!!!

Announcements

- Your Lab 4 is meeting normally this week!
 - Make sure you attend your correct section
- Homework 3 is out
 - Due by Thursday (Sept 24th) at 8:59:59 PM
- Homeworks are on Blackboard
 - Weekly Agendas are also on Blackboard