Lecture 11:
Iteration and For-Loops
(Sections 4.2 and 10.3)
CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Fan, D. Gries, L. Lee,
S. Marschner, C. Van Loan, W. White]
Announcements

• Be sure to monitor email for course announcements
• A2 due Mar 19 at 11:59pm
• Window to submit A1 revisions closes Mar 20 at 11:59pm
Important concept in computing: Doing things *repeatedly*

1. Perform $n$ trials or get $n$ samples.
   - Run a protein-folding simulation for $10^6$ time steps
   - Next 50 ticket purchases entered in random draw for upgrade

2. Process each item in a sequence
   - Compute aggregate statistics (e.g., mean, median) on scores
   - Send everyone in a Facebook group an appointment time

3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal

Repeat a known *(definite)* number of times

Repeat until something happens—repeat an *indefinite* number of times
1st Attempt: Summing the Elements of a List

```python
def sum(the_list):
    """Returns: the sum of all elements in the_list
    Precondition: the_list is a list of all numbers (either floats or ints)"""
    result = 0
    result = result + the_list[0]
    result = result + the_list[1]
    ...  # Houston, we have a problem
    return result
```
Working with Sequences

• Sequences are potentially **unbounded**
  ▪ Number of elements is not fixed
  ▪ Functions must handle sequences of different lengths
  ▪ **Example:** \( \text{sum}([1,2,3]) \) vs. \( \text{sum}([4,5,6,7,8,9,10]) \)

• Cannot process with **fixed** number of lines
  ▪ Each line of code can handle at most one element
  ▪ What if there are millions of elements?

• We need a new approach
For Loops: Processing Sequences

```python
for x in grades:
    print(x)
```

- **loop sequence**: grades
- **loop variable**: x
- **loop body**: `print(x)`

To execute the for-loop:

1) Check if there is a “next” element of **loop sequence**

2) If so:
   - assign next sequence element to **loop variable**
   - Execute all of **the body**
   - Go back to 1)

3) If not, terminate execution
def sum(the_list):
    """Returns: the sum of all elements in the_list
Precondition: the_list is a list of all numbers
(either floats or ints)"
    result = 0
    for x in the_list:
        result = result + x
    return result

• loop sequence: the_list
• loop variable: x
• body: result=result+x
def num_zeroes(the_list):
    """Returns: the number of zeroes in the_list
    Precondition: the_list is a list"""
    count = 0  # Create var. to keep track of 0's
    for x in the_list:  # for each element in the list...
        if x == 0:  # check if it is equal to 0
            count = count + 1  # add 1 if it is
    return count  # Return the variable/counter
def num_zeroes(the_list):
    """Returns: the number of zeroes in the_list
    Precondition: the_list is a list"""
    count = 0
    for x in the_list:
        if x == 0:
            count = count + 1
    return count
Accumulator

• A variable to hold a final answer
• for-loop adds to the variable at each step
• The final answer is accumulated, i.e., built up, one step at a time. A common design *pattern*:

```plaintext
accumulator
for ________:
    accumulator = accumulator + ________
```

• Accumulator does not need to be a number. E.g., can be a string to be built-up
def ave_positives(my_list):
    """Returns: average (float) of the positive values in my_list
    my_list: a list of numbers with at least one positive value"""

• Be goal oriented → can work backwards
• Name a variable for any value that you need but don’t have yet
• Break down a problem!
  • ... break into parts
  • ... solve simpler version first
• Remember loop/accumulation pattern
What if we aren’t dealing with a list?

So far we’ve been building for-loops around elements of a list.

What if we just want to do something some number of times?

`range` to the rescue!
**range: a handy counting function!**

- **range(x)**
  - Generates 0, 1, ..., x-1

- **Important:** range does not return a list
  - Can convert range’s return value into a list

- **range(a, b)**
  - Generates a, ..., b-1

- **range(a, b, s)**
  - Generates a, a+s, a+2s, ..., b-1

---

```python
>>> print(range(6))
range(0, 6)

>>> first_six = list(range(6))
>>> print(first_six)
[0, 1, 2, 3, 4, 5]

>>> second_six = list(range(6, 13))
>>> print(second_six)
[6, 7, 8, 9, 10, 11, 12]
```
Modifying the Contents of a List

```python
def add_bonus(grades):
    """Adds 1 to every element in a list of grades (either floats or ints)""
    size = len(grades)
    for k in range(size):
        grades[k] = grades[k]+1

lab_scores = [8,9,10,5,9,10]
print("Initial grades are: "+str(lab_scores))
add_bonus(lab_scores)
print("With bonus, grades are: "+str(lab_scores))
```

If you need to modify the list, you need to use `range` to get the indices.

Watch this in the python tutor!
Common For-Loop Mistakes

Mistake #1: Modifying the loop variable instead of the list itself.

Mistake #2: Modifying the loop sequence as you walk through it.
For-Loop Mistake #1 (Q)

Modifying the loop variable (here: x).

```python
def add_one(the_list):
    """Adds 1 to every element in the list
    Precondition: the_list is a list of all numbers
    (either floats or ints)"
    for x in the_list:
        x = x + 1

a = [5, 4, 7]
add_one(a)
print(a)
```

What gets printed?

A: [5, 4, 7]
B: [5, 4, 7, 5, 4, 7]
C: [6, 5, 8]
D: Error
E: I don’t know
def add_one(the_list):
    """Adds 1 to every elt
    Pre: the_list is all numb.""
    for x in the_list:
        x = x + 1

grades = [5, 4, 7]
add_one(grades)
For-Loop Mistake #2 (Q)

Modifying the loop sequence as you walk through it.

What gets printed?

b = [1, 2, 3]
for a in b:
    b.append(a)
print(b)

A: never prints b
B: [1, 2, 3, 1, 2, 3]
C: [1, 2, 3]
D: I do not know