Lecture 11: Iteration and For-Loops
(Sections 4.2 and 10.3)

CS 1110
Introduction to Computing Using Python

http://www.cs.cornell.edu/courses/cs1110/2022sp

Important concept in computing: Doing things \textit{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

4. Repeat a known (definite) number of times
   - Repeat until something happens
   - Repeat an indefinite number of times

\textit{1st Attempt: Summing the Elements of a List}

\begin{verbatim}
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]
    ... return result
\end{verbatim}

\textit{Working with Sequences}

- Sequences are potentially \textbf{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 \textbf{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

\textit{For Loops: Processing Sequences}

\begin{verbatim}
for x in grades:
    print(x)
\end{verbatim}

- \textbf{loop sequence}: grades
- \textbf{loop variable}: \(x\)
- \textbf{loop body}: \(\text{print}(x)\)

To execute the for-loop:
1) Check if there is a “next” element of \textit{loop sequence}
2) If so:
   - assign next sequence element to \textit{loop variable}
   - Execute all of \textit{the body}
   - Go back to 1)
3) If not, terminate execution

Houston, we have a problem
Solution: 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
    for x in the_list:
        result = result + x

    return result
```

For Loops and Conditionals

```python
def num_zeros(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:
        if x == 0:
            count = count + 1  # add 1 if it is 0
    return count  # Return the variable/counter
```

Exercise

```python
def ave_positives(my_list):
    """Returns: avg (float) of positive values in my_list
    my_list: a list of #s with at least 1 positive value"
```

For Loop with labels

```python
def num_zeros(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:

```
accumulator
for ________:
    accumulator = accumulator + ______
```

- Accumulator does not need to be a number. E.g., can be a string to be built-up

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!

\[
\begin{align*}
\text{range}(x) & \quad \text{generates } 0, 1, \ldots, x-1 \\
\text{Important: range does not return a list} & \quad \text{can convert range's return value into a list}
\end{align*}
\]

Arguments must be int expressions

\[
\begin{align*}
\text{range}(a,b) & \quad \rightarrow a, \ldots, b-1 \\
\text{range}(a,b,s) & \quad \rightarrow a, a+s, a+2s, \ldots, b-1
\end{align*}
\]

What gets printed? (Q)

\[
t = 0 \\
\text{for } k \text{ in range}(5, 1, -1): \\
\quad t = t + 1 \\
\text{print}(t)
\]

\[
\begin{array}{c}
A: 0 \\
B: 2 \\
C: 3 \\
D: 4 \\
E: 5
\end{array}
\]

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))
```

Watch this in the python tutor!

Common For-Loop Mistake #1

Modifying the loop variable instead of the list itself.

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
```

What gets printed?

\[
\begin{array}{c}
A: [5, 4, 7] \\
B: [5, 4, 7, 5, 4, 7] \\
C: [6, 5, 8] \\
D: Error \\
E: I don't know
\end{array}
\]

Modifying the Loop Variable (1)

```python
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)
```
Modifying the Loop Variable (2)

```python
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)
```

Modifying the Loop Variable (3)

```python
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)
```

Modifying the Loop Variable (4)

```python
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)
```

Modifying the Loop Variable (5)

```python
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)
```

Modifying the Loop Variable (6)

```python
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)
```

Modifying the Loop Variable (7)

```python
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)
```
Modifying the Loop Variable (8)

```python
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)
```

Modifying the Loop Variable (9)

```python
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)
```

Common For-Loop Mistakes #2

Modifying the loop sequence as you walk through it.

For-Loop Mistake #2 (Q)

Modifying the loop sequence as you walk through it.

What gets printed?

```python
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