Lecture 13

For-Loops
Announcements for This Lecture

Reading

- Today: Chapters 8, 10
- Thursday: Chapter 11

Assignments/Lab

- A2 has been graded
  - Pick up in Gates 216
  - Grades generally good
- A3 is due on Thursday
  - Will post survey today
  - Survey due next week
- Lab is on lists/for-loops
  - Due in two weeks
  - But fair game on exam

Prelim, Oct 12th 7:30-9:00

- Material up to TODAY
- Study guide is posted

Review next Wednesday

- Room/Time are TBA
- Will cover what is on exam

10/3/17

For Loops
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"
    pass  # Stub to be implemented
```

Remember our approach:
Outline first; then implement
Example: Summing the Elements of a List

def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"""
    # Create a variable to hold result (start at 0)
    # Add each list element to variable
    # Return the variable
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)""
    result = 0
    result = result + thelist[0]
    result = result + thelist[1]
    ...
    return result
```

There is a problem here
Working with Sequences

• Sequences are potentially **unbounded**
  ▪ Number of elements inside them is not fixed
  ▪ Functions must handle sequences of different lengths
  ▪ **Example**: `sum([1,2,3])` vs. `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 # of elements > # of lines of code?

• We need a new **control structure**
# Print contents of `seq`
x = seq[0]
print(x)
x = seq[1]
print(x)
...
x = seq[len(seq)-1]
print(x)

- **Remember:**
  - Cannot program …

**The for-loop:**
```python
for x in seq:
    print(x)
```

**Key Concepts**
- **loop sequence:** `seq`
- **loop variable:** `x`
- **body:** `print(x)`
- Also called **repetend**
The for-loop:
```
for x in seq:
    print(x)
```

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

To execute the for-loop:
1. Check if there is a “next” element of **loop sequence**
2. If not, terminate execution
3. Otherwise, put the element in the **loop variable**
4. Execute all of **the body**
5. Repeat as long as 1 is true
Example: Summing the Elements of a List

def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers
    (either floats or ints)""
    # Create a variable to hold result (start at 0)
    # Add each list element to variable
    # Return the variable
def sum(thelist):
    
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"

    result = 0

    for x in thelist:
        result = result + x

    return result
Example: Summing the Elements of a List

```python
def sum(thelist):
    """Returns: the sum of all elements in thelist
    Precondition: thelist is a list of all numbers (either floats or ints)"
    result = 0
    for x in thelist:
        result = result + x
    return result
```

Accumulator variable

- loop sequence: thelist
- loop variable: x
- body: result = result + x
def num_ints(thelist):
    """Returns: the number of ints in thelist
    Precondition: thelist is a list of any mix of types""
    # Create a variable to hold result (start at 0)
    # for each element in the list...
    # check if it is an int
    # add 1 if it is
    # Return the variable
def num_ints(thelist):
    """Returns: the number of ints in thelist
    Precondition: thelist is a list of any mix of types""
    result = 0
    for x in thelist:
        if type(x) == int:
            result = result + 1
    return result
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers (either floats or ints)"

    for x in thelist:
        x = x + 1

    # procedure; no return

    DOES NOT WORK!
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):
For Loops and Call Frames

```python
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):
```

```
1  for x in thelist:
2      x = x + 1
```

```
10/3/17
```

```
add_one
thelist  id4
  x  5
```

```
seq  id4
  0  5
  1  4
  2  7
```

```
For Loops
  16
```

10/3/17
For Loops and Call Frames

```python
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    for x in thelist:
        x = x + 1
```

```
add_one(seq):
```

```
for x in thelist:
    x = x + 1
```

Loop back to line 1

Increments x in frame
Does not affect folder
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x+1

add_one(seq):

seq

<table>
<thead>
<tr>
<th>id4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

id4

<table>
<thead>
<tr>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

thelist

<table>
<thead>
<tr>
<th>id4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

x

| 4 |

Next element stored in x.
Previous calculation lost.
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1
def add_one(thelist):
    
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    
    for x in thelist:
        x = x + 1

add_one(seq):

seq

id4

0 5
1 4
2 7

add_one
2

thelist
id4
x 7

Next element stored in x.
Previous calculation lost.
For Loops and Call Frames

```python
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):
```

```
seq
  id4
0  5
1  4
2  7
```

```
add_one
  thelist  id4
    x     8
```

Loop back to line 1
def add_one(thelist):
    '''Adds 1 to every elt
    Pre: thelist is all numb.'''
    for x in thelist:
        x = x + 1

add_one(seq):

Loop is completed.
Nothing new put in x.
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):

seq  id4
   0  5
   1  4
   2  7

No changes to folder
On The Other Hand

```python
def copy_add_one(thelist):
    """Returns: copy with 1 added to every element
    Precondition: thelist is a list of all numbers (either floats or ints)"""
    mycopy = []  # accumulator
    for x in thelist:
        x = x + 1
        mycopy.append(x)  # add to end of accumulator
    return mycopy
```

Accumulator keeps result from being lost
How Can We Modify A List?

- **Never** modify loop var!
- This is an infinite loop:

```python
for x in thelist:
    thelist.append(1)
```

- Need a second sequence
- How about the *positions*?

```python
thelist = [5, 2, 7, 1]
thepos = [0, 1, 2, 3]
for x in thepos:
    thelist[x] = thelist[x] + 1
```

Try this in Python Tutor to see what happens
How Can We Modify A List?

- **Never** modify loop var!
- This is an infinite loop:

  ```python
  for x in thelist:
      thelist.append(1)
  ```

- Need a second sequence
- How about the *positions*?

  ```python
  thelist = [5, 2, 7, 1]
  thepos = [0, 1, 2, 3]
  for x in thepos:
      thelist[x] = thelist[x] + 1
  ```

Try this in Python Tutor to see what happens.
This is the Motivation for Iterators

- Iterators are objects
  - Contain data like a list
  - But cannot slice them
- Access data with `next()`
  - Function to get next value
  - Keeps going until end
  - Get an error if go too far
- Can convert back & forth
  - `myiter = iter(mylist)`
  - `mylist = list(myiter)`
Iterators and Lists

```python
>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> next(alt)
5
>>> next(alt)
4
>>> next(alt)
7
>>> next(alt)
Traceback...
```
Iterators and For Loops

>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> for x in alt:
    print(x)
5
4
7

Just like looping over the list
Iterators and For Loops

```python
>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> for x in alt:
    print(x)
5
4
7
```

But still not safe to modify iterator’s list

Just like looping over the list
The Range Iterator

- **range(x)**
  - Creates an iterator
  - Stores $[0, 1, \ldots, x-1]$  
  - **But not a list!**
  - But try `list(range(x))`
- **range(a,b)**
  - Stores $[a, \ldots, b-1]$
- **range(a,b,n)**
  - Stores $[a, a+n, \ldots, b-1]$

- Very versatile tool
- Great for processing ints

Accumulator

```
total = 0
# add the squares of ints # in range 2..200 to total
for x in range(2,201):
total = total + x*x
```
def add_one(thelist):
    """(Procedure) Adds 1 to every element in the list
    Precondition: thelist is a list of all numbers (either floats or ints)""

    size = len(thelist)
    for k in range(size):
        thelist[k] = thelist[k]+1
    # procedure; no return
Important Concept in CS: Doing Things Repeatedly

1. Process each item in a sequence
   - Compute aggregate statistics for a dataset, such as the mean, median, standard deviation, etc.
   - Send everyone in a Facebook group an appointment time

2. Perform $n$ trials or get $n$ samples.
   - A4: draw a triangle six times to make a hexagon
   - Run a protein-folding simulation for $10^6$ time steps

3. Do something an unknown number of times
   - CUAUV team, vehicle keeps moving until reached its goal
Important Concept in CS: Doing Things Repeatedly

1. Process each item in a sequence
   - Compute aggregate statistics for a dataset, such as the mean, median, standard deviation, etc.
   - Send everyone in a Facebook group an appointment time

2. Perform $n$ trials or get $n$ samples.
   - A4: draw a triangle six times to make a hexagon
   - Run a protein-folding simulation for $10^6$ time steps

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

For Loops

```
for x in sequence:
    process x
```

```
for x in range(n):
    do next thing
```

Cannot do this yet
Impossible w/ Python for

10/3/17

For Loops