CS 1110: Introduction to Computing Using Python

Lecture 11

Iteration and For Loops

[Andersen, Gries, Lee, Marschner, Van Loan, White]
Announcements: Prelim 1

- Rooms:
  - aa200 – jjm200  Baker Laboratory 200
  - jjm201 – sge200  Rockefeller 201
  - sge201 – zz200  Rockefeller 203

- covers material up through today
  no assert, try-except

- What to study: A1, A2, Labs 1-6, old exam questions:
  - Fall 2016, 2015, 2014 call-frame/diagram questions need to be converted to our notation.

- Prelim will probably be closer in style to Spring 2013-2014 than more recent exams
Prelim 1: Things that are not “fair game”

- Prelim 1 fall 2016: ignore 3b (too lecture-dependent)
- Prelim 1 spring 2016: ignore 1, 3, 6.
  - 4 is OK if you ignore the "if name == ..." line, and just assume all that stuff is script code to be run
- Prelim 1 fall 2015: ignore 4(a) – solutions have typos
  - 4(c) not fair game (asserts)
- Prelim 1 spring 2015: ignore 2(b), 3(b), 5
  - For 1(b), imagine that variable s contains some arbitrary, unknown string (we didn't formally cover raw_input)
- Prelim 1 fall 2014: ignore 2(e), 4(a)
- Prelim 1 spring 2013: question 6: change cunittest2 to cornelltest
More Announcements

• A2: due today. Solutions released Thursday.
• Lab 6: due in two weeks
  ▪ Tuesday 3/14 labs: open office hours
  ▪ Wednesday 3/15 labs: cancelled
• Thursday 3/9: optional in-class review session
• Tuesday 3/14: no lecture; office hours instead
  ▪ Olin 155 during class times, Carpenter in between
• A3: released sometime after Prelim 1
Tuples

- Tuples fall between strings and lists
  - write them with just commas: 42, 4.0, ‘x’
  - often enclosed in parentheses: (42, 4.0, ‘x’)

Conventionally use lists for:
- long sequences
- homogeneous sequences
- variable length sequences

Conventionally use tuples for:
- short sequences
- heterogeneous sequences
- fixed length sequences

strings: immutable sequences of characters

lists: mutable sequences of any objects

“tuple” generalizes “pair,” “triple,” “quadruple,” …

tuples: immutable sequences of any objects
Returning multiple values

- Can use lists/tuples to return multiple values

```python
def div_rem(x, y):
    return (x / y, x % y)

>>> div_rem(3, 2)
(1, 1)
```
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)"""
```

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 approach
The Map Function

- **map**($\langle\text{function}\rangle, \langle\text{list}\rangle$)
  - Function has to have exactly 1 parameter
  - Otherwise, get an error
  - Returns a new list

\[ \text{map}(\text{f}, x) \]

\[ [f(x[0]), f(x[1]), \ldots, f(x[n-1])] \]

calls the function \(f\) once for each item

\[
\text{map}(\text{len}, [\text{'a'}, \text{'bc'}, \text{'defg'}])
\]

returns \([1, 2, 4]\)
The Filter Function

- `filter(⟨Boolean_function⟩, ⟨list⟩)`
  - Function must:
    - have exactly 1 parameter
    - return a Boolean
  - Returns a new list
- Returns elements of ⟨list⟩ for which ⟨Boolean_function⟩, returns True

```
filter(f, x) = [f(x[0]), f(x[1]), ..., f(x[n-1])]
```

calls the function f once for each item
For Loops: Processing Sequences

# Print contents of seq
x = seq[0]
print x
x = seq[1]
print x
...
x = seq[len(seq)-1]
print x

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:

```python
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, **assign** element to 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
What gets printed?

```python
a = 0
for b in [1]:
    a = a + 1

print a
```

prints 1
What gets printed?

\[
\begin{align*}
\text{a} &= 0 \\
\text{for } \text{b in [1, 2]}: \\
\quad \text{a} &= \text{a} + 1 \\
\end{align*}
\]

prints 2

\[
\text{print a}
\]
What gets printed?

```python
a = 0
for b in [1, 2, 3]:
    a = a + 1
print a
```

prints 3

print a
What gets printed?

```python
a = 0
for b in [1, 2, 3]:
    a = b
prints 3
print a
```

3/7/17 Iteration and For Loops 20
What gets printed?

\[
\begin{align*}
a &= 0 \\
\text{for } b \text{ in } [1, 2, 3]: \\
&\quad a = a + b
\end{align*}
\] 

print \ a

prints 6
What gets printed?

```python
a = 0
b = [1, 2, 3]
for c in b:
    a = a + c
```

print a

prints 6
What gets printed?

\[\begin{align*}
a &= 0 \\
b &= [1, 2, 3] \\
\text{for } c \text{ in } b: & \quad \text{prints } [1, 2, 3] \\
\quad a &= a + c \\
\text{print } b
\end{align*}\]
What gets printed?

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

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

* Runs out of memory eventually, then probably throws an error.
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

    sounds kind of like filter
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
Modifying the Contents of a List

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

>>> a = [5, 4, 7]
>>> add_one(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(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
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 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
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb.""
    for x in thelist:
        x = x + 1

for x in seq:
    x = x + 1

for x in thelist:
    add_one(x)
Modifying the Contents of a List

```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):

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

add_one(seq):

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

Modifying the Contents of a List

```python
add_one(seq):
```

```
add_one
thelist id4
x 8
```

```
Loop is completed.
Nothing new put in x.
```

seq id4

```
0 5
1 4
2 7
```

3/7/17
def add_one(thelist):
    """Adds 1 to every elt
    Pre: thelist is all numb."""
    for x in thelist:
        x = x + 1

add_one(seq):

No changes to folder
Modifying the Contents of a List

```python
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

>>> a = [1, 2, 3]
>>> add_one(a)
>>> a
```

**What gets printed?**

- **A**: [1, 2, 3]  **CORRECT**
- **B**: [1, 2, 3, 1, 2, 3]
- **C**: [2, 3, 4]
- **D**: Error
- **E**: I don’t know
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

>>> a = [1, 2, 3]
>>> add_one(a)

>>> a
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
Range Function

- \texttt{range(x)}: returns a list of ints from 0 to \(x-1\)
- \texttt{range(a,b)}: returns a list of ints from \(a\) to \(b-1\)
For each x in the range 2..200, add x*x to total

```python
total = 0
for x in range(2, 201):
    total = total + x*x
```

# add the squares of ints # in range 2..200 to total

```python
total = total + 2*2
total = total + 3*3
...
total = total + 200*200
```
What gets printed?

```
a = 0
for b in range(0, 1):
    a = a + 1
print a
```

prints 1

```
print a
```
What gets printed?

```
a = 0
for b in range(0, 4):
    a = a + 1
print a
```

prints 4

```
print a
```
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