

CS 1110:

Introduction to Computing Using Python

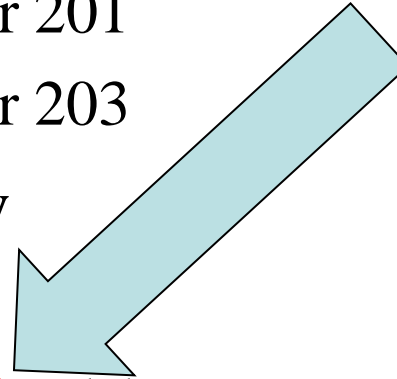
Lecture 11

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



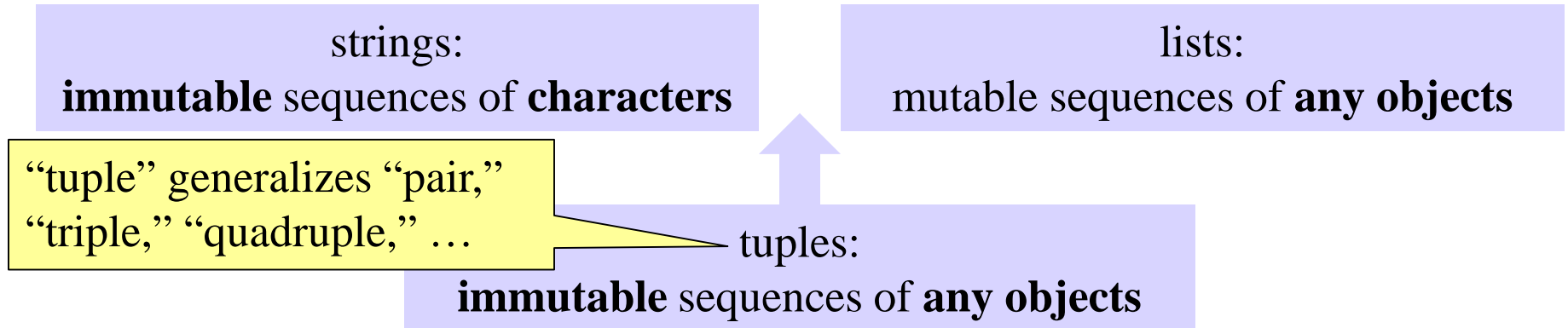
Prelim 1: Things that are not “fair game”

- 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

Returning multiple values

- Can use lists/tuples to **return** multiple values

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

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

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)"""
```

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

```
def sum(thelist):
```

```
    """Returns: the sum of all elements in thelist  
    Precondition: thelist is a list of all numbers  
    (either floats or ints)"""
```

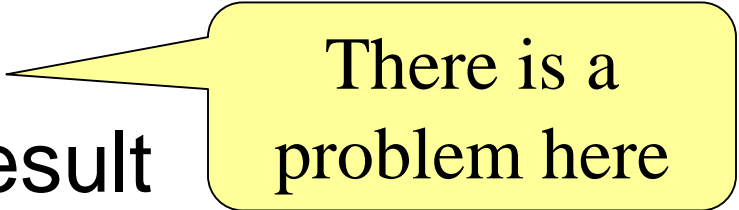
```
    result = 0
```

```
    result = result + thelist[0]
```

```
    result = result + thelist[1]
```

```
    ...
```

```
    return result
```



There is a
problem here

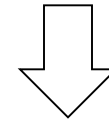
Working with Sequences

- 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(function, list)`
 - Function has to have exactly **1 parameter**
 - Otherwise, get an error
 - Returns a new list

`map(f, x)`



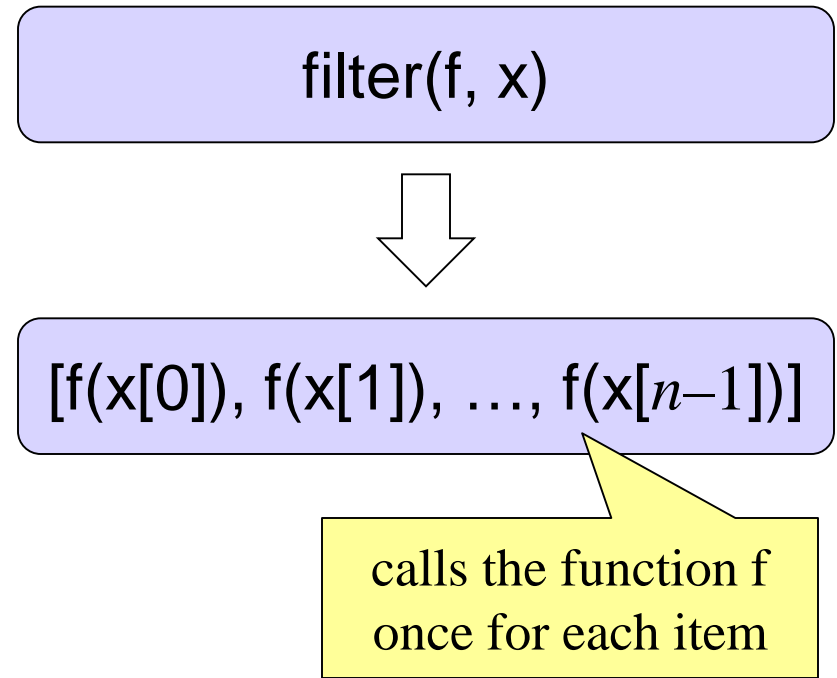
`[f(x[0]), f(x[1]), ..., f(x[n-1])]`

calls the function `f`
once for each item

`map(len, ['a', 'bc', '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



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:

```
for x in seq:
    print x
```

- Key Concepts
 - **loop sequence:** seq
 - **loop variable:** x
 - **body:** print x
 - Also called **repetend**

For Loops

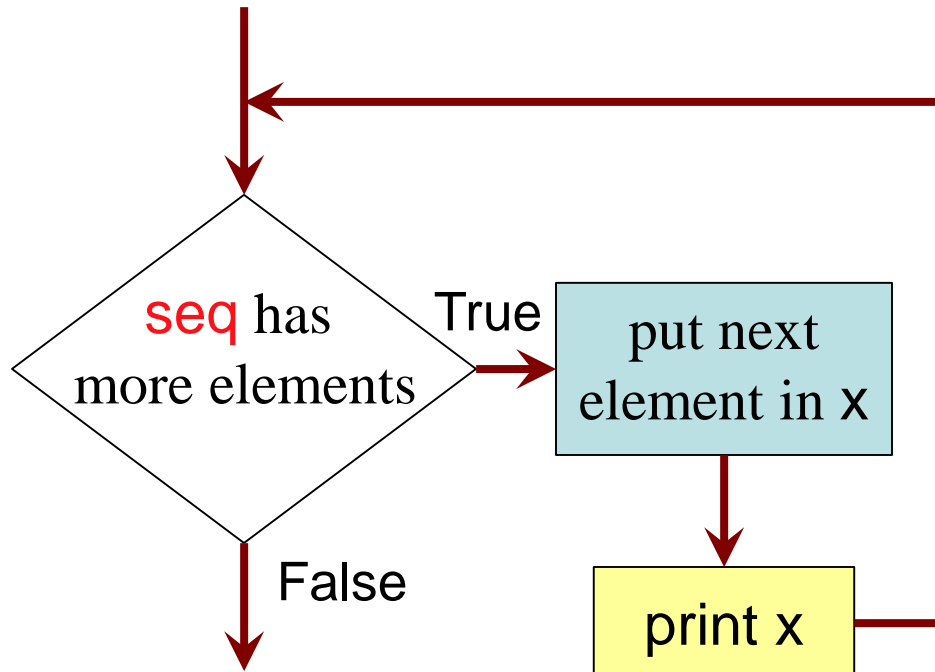
The for-loop:

```
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, *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
```

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)"""
```

```
    result = 0
```

Accumulator
variable

```
    for x in thelist:
```

```
        result = result + x
```

```
    return result
```

- **loop sequence:** thelist
- **loop variable:** x
- **body:** result=result+x

What gets printed?

```
a = 0
```

```
for b in [1]:
```

```
    a = a + 1
```

prints 1

```
print a
```

What gets printed?

```
a = 0
```

```
for b in [1, 2]:
```

```
    a = a + 1
```

prints 2

```
print a
```

What gets printed?

```
a = 0
```

```
for b in [1, 2, 3]:
```

```
    a = a + 1
```

prints 3

```
print a
```

What gets printed?

```
a = 0
```

```
for b in [1, 2, 3]:
```

```
    a = b
```

prints 3

```
print a
```

What gets printed?

```
a = 0
```

```
for b in [1, 2, 3]:
```

```
    a = a + b
```

prints 6

```
print a
```

What gets printed?

```
a = 0
```

```
b = [1, 2, 3]
```

```
for c in b:
```

```
    a = a + c
```

prints 6

```
print a
```

What gets printed?

```
a = 0
```

```
b = [1, 2, 3]
```

```
for c in b:
```

```
    a = a + c
```

prints [1, 2, 3]

```
print b
```

What gets printed?

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

INFINITE LOOP!

```
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.**

For Loops and Conditionals

```
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****

For Loops and Conditionals

```
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
```



Body

```
    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)
```

```
>>> a
```

What gets printed?

A: [5, 4, 7]

B: [5, 4, 7, 5, 4, 7]

C: [6, 5, 8]

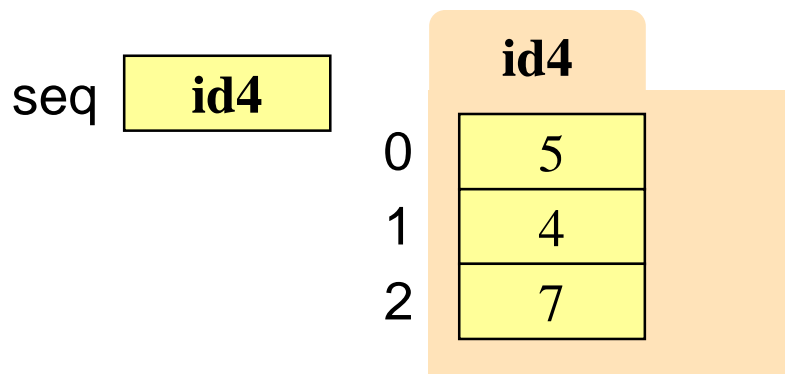
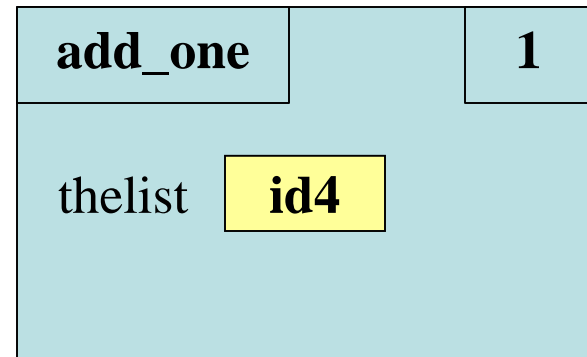
D: **Error**

E: I don't know

Modifying the Contents of a List

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

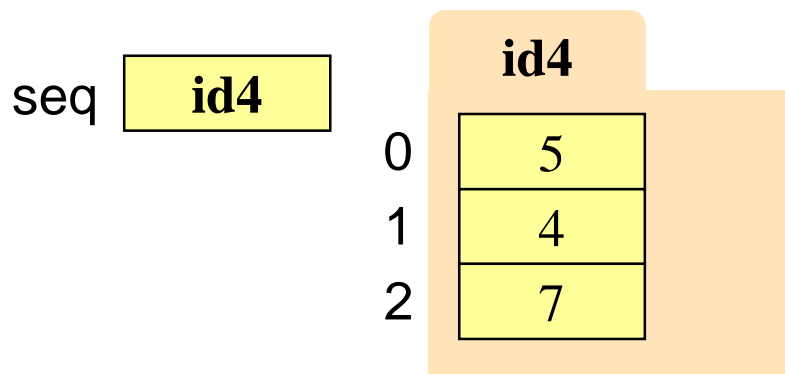
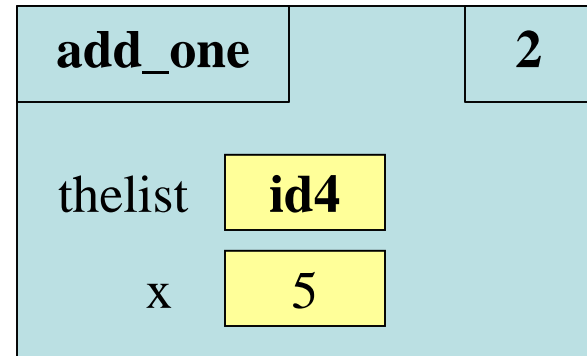
add_one(seq):



Modifying the Contents of a List

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

add_one(seq):

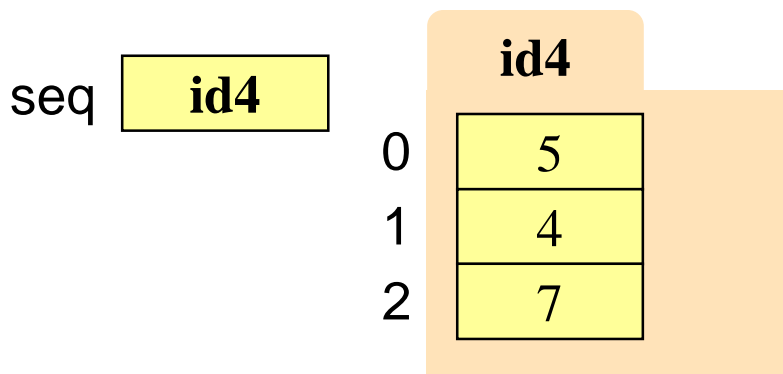
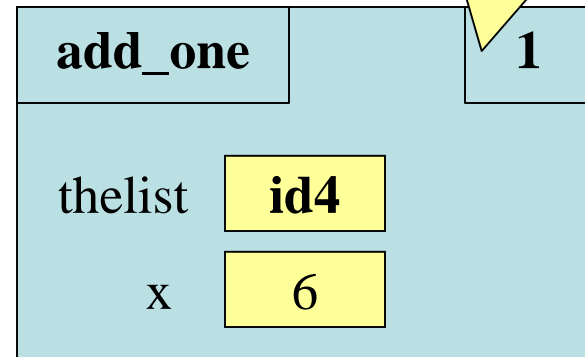


Modifying the Contents of a List

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

add_one(seq):

Loop back
to line 1

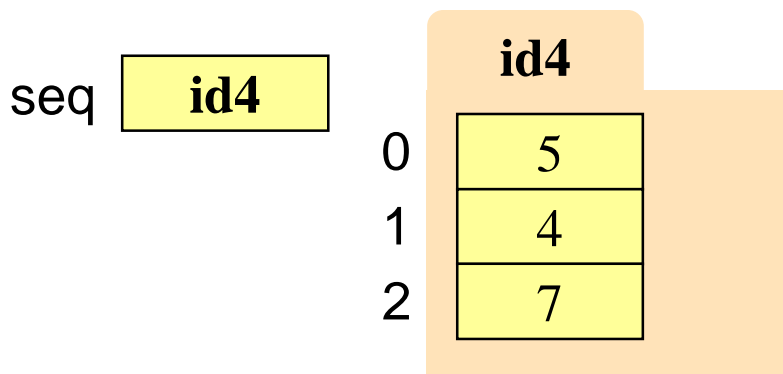
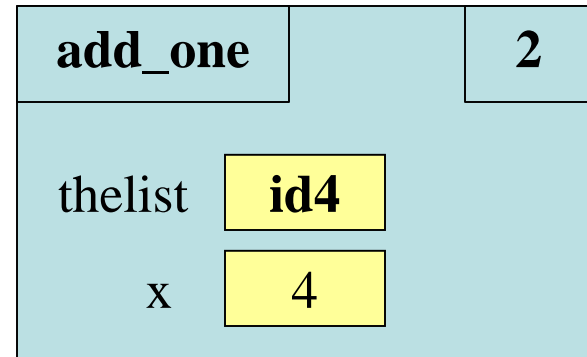


Increments x in **frame**
Does not affect folder

Modifying the Contents of a List

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

add_one(seq):



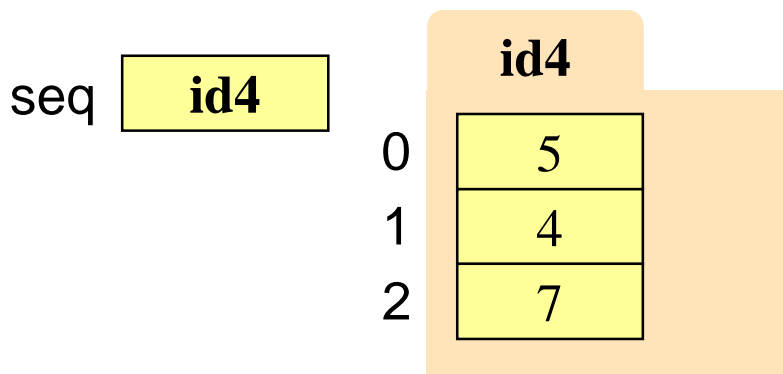
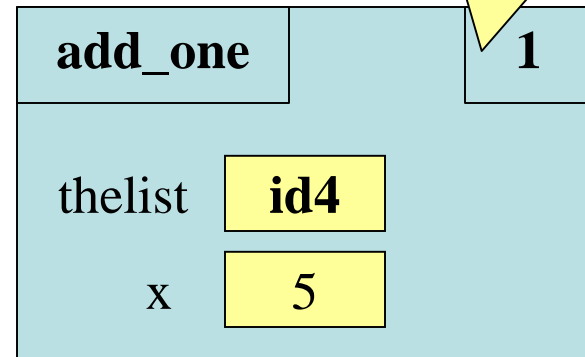
Next element stored in x.
Previous calculation lost.

Modifying the Contents of a List

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

add_one(seq):

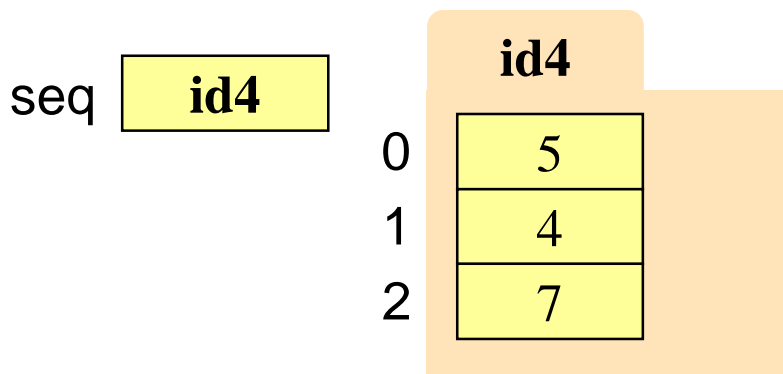
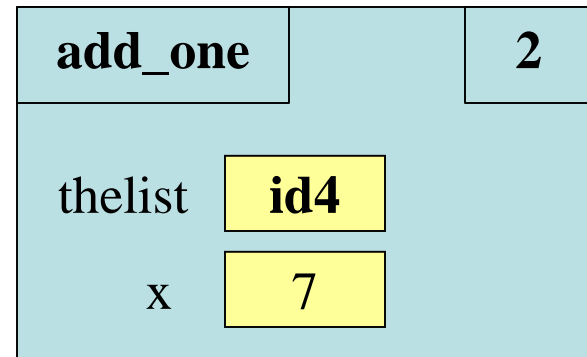
Loop back
to line 1



Modifying the Contents of a List

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

add_one(seq):



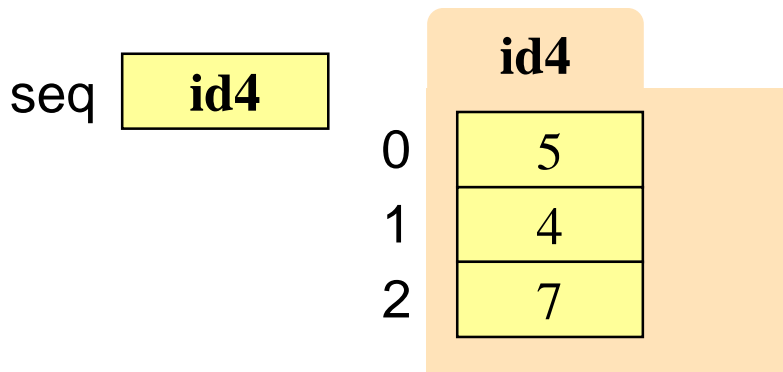
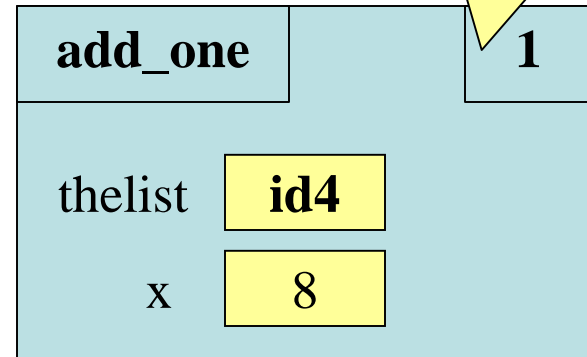
Next element stored in x.
Previous calculation lost.

Modifying the Contents of a List

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

add_one(seq):

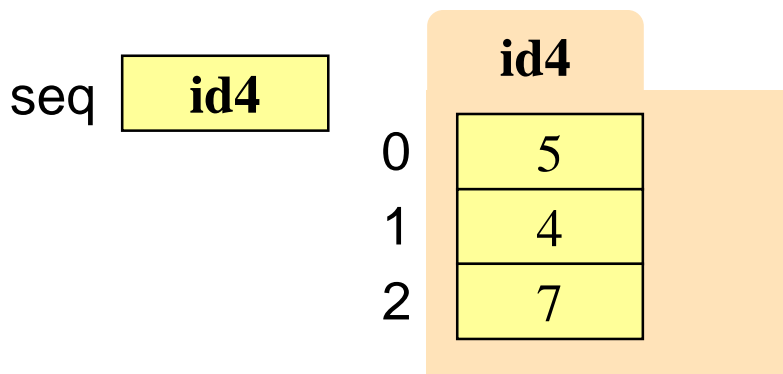
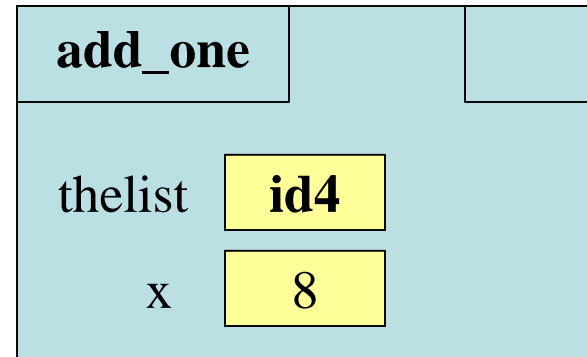
Loop back
to line 1



Modifying the Contents of a List

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

add_one(seq):



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

Modifying the Contents of a List

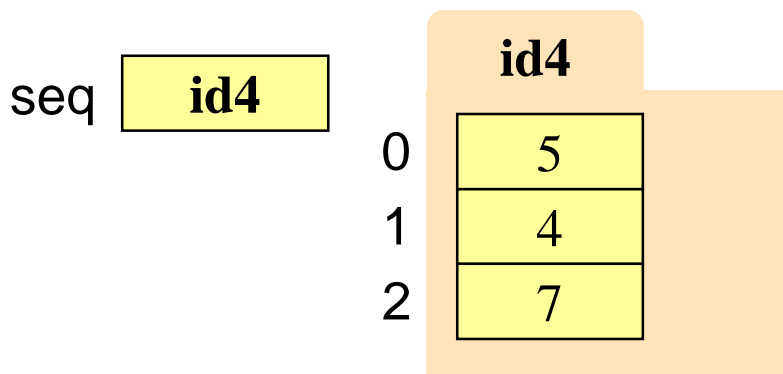
```
def add_one(thelist):          add_one(seq):
```

```
    """Adds 1 to every elt  
    Pre: thelist is all numb."""
```

```
1 for x in thelist:
```

```
2     x = x+1
```

ERASE WHOLE FRAME



No changes
to folder

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 = [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

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
```

DOES NOT WORK!

```
>>> a = [1, 2, 3]
```

```
>>> add_one(a)
```

```
>>> a
```

On The Other Hand

```
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)"""
```

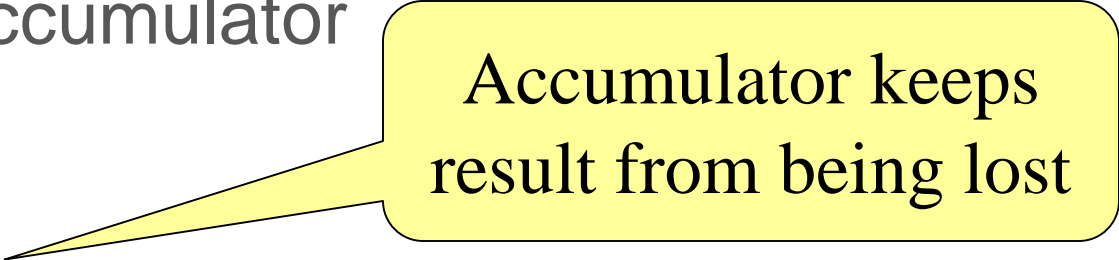
```
    mycopy = [] # accumulator
```

```
    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

- `range(x)`:
returns a list of ints from 0 to $x-1$
- `range(a,b)`:
returns a list of ints from a to $b-1$

For Loops: Processing Ranges of Integers

- For each x in the range 2..200, add $x*x$ to total

total = 0

add the squares of ints

in range 2..200 to total

total = total + 2*2

total = total + 3*3

...

total = total + 200*200

```
total = 0
```

```
for x in range(2,201):
```

```
    total = total + x*x
```

What gets printed?

```
a = 0
```

```
for b in range(0, 1):
```

```
    a = a + 1
```

prints 1

```
print a
```

What gets printed?

```
a = 0
```

```
for b in range(0, 4):
```

```
    a = a + 1
```

prints 4

```
print a
```

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)"""
```

```
    size = len(thelist)
```

```
    for k in range(size):
```

```
        thelist[k] = thelist[k]+1
```

```
    # procedure; no return
```



WORKS!