

## Lecture 13

# For-Loops

# Announcements for This Lecture

---

## Reading

---

- Today: Chapters 8, 10
- Thursday: Chapter 11
- **Prelim, Oct 12<sup>th</sup> 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

## 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

# 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)"""
```

```
    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

---

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

```
    """Returns: the sum of all elements in thelist
```

```
    Precondition: thelist is a list of all numbers  
    (either floats or ints)"""
```

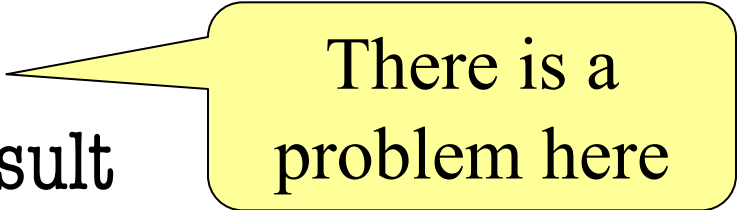
```
    result = 0
```

```
    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**

# 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)
```

- **Remember:**
  - Cannot program ...

## The for-loop:

```
for x in seq:
    print(x)
```

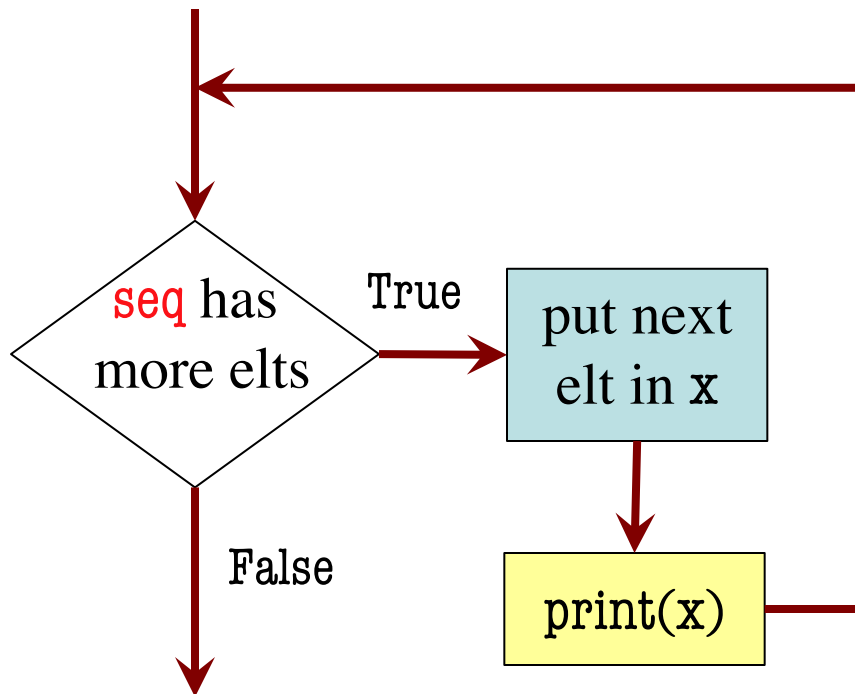
- Key Concepts
  - **loop sequence:** seq
  - **loop variable:** x
  - **body:** print(x)
  - Also called **repetend**

# For Loops: Processing Sequences

**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
```

# 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
```

```
    for x in thelist:
```

```
        result = result + x
```

```
    return result
```

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

# 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

# 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
```

# 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
```

```
    # procedure; no return
```

**DOES NOT WORK!**

# For Loops and Call Frames

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

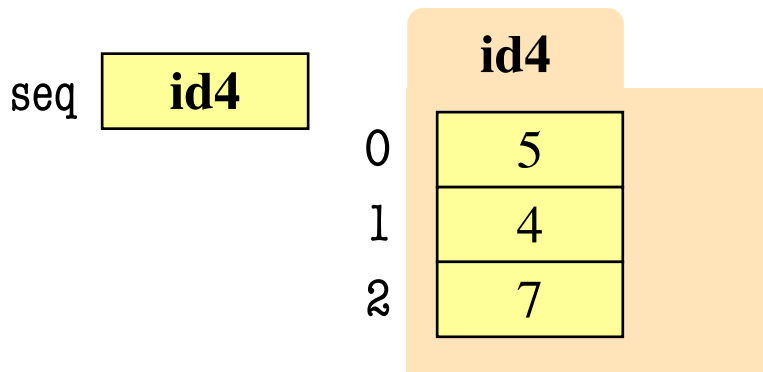
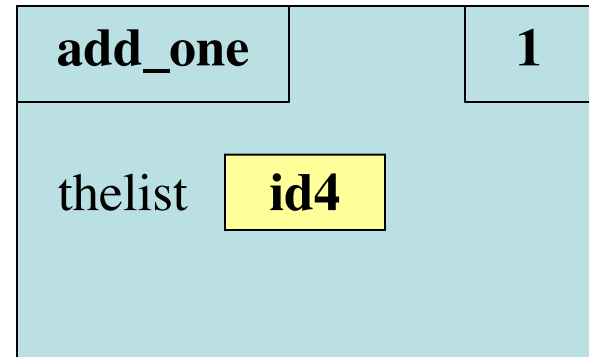
```
    """Adds 1 to every elt
```

```
    Pre: thelist is all numb."""
```

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

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

```
add_one(seq):
```



# For Loops and Call Frames

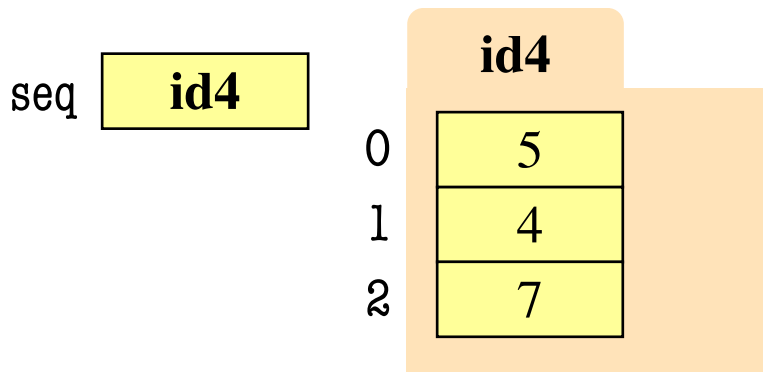
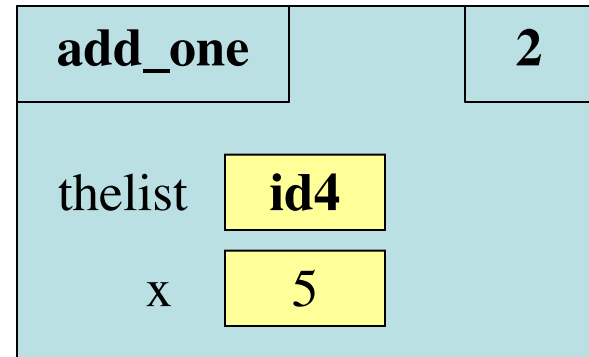
```
def add_one(thelist):
```

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

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

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

```
add_one(seq):
```



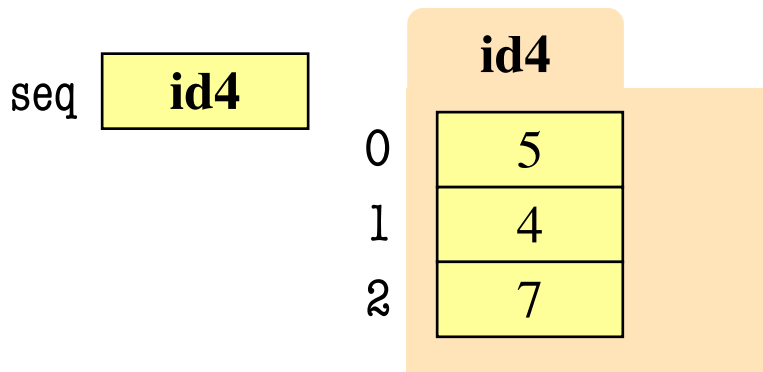
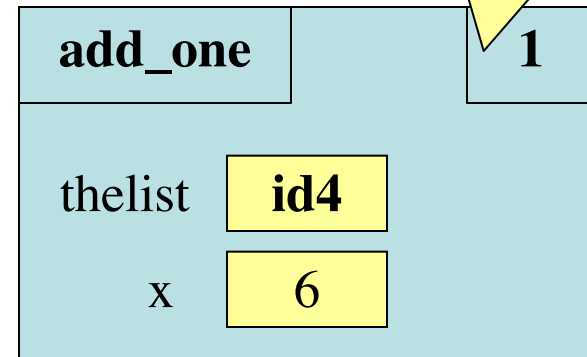


# For Loops and Call Frames

```
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

# For Loops and Call Frames

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

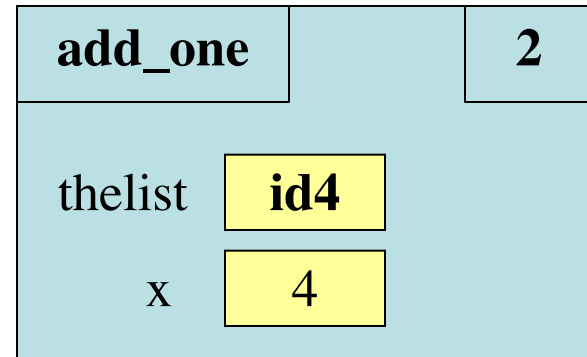
```
    """Adds 1 to every elt
```

```
    Pre: thelist is all numb."""
```

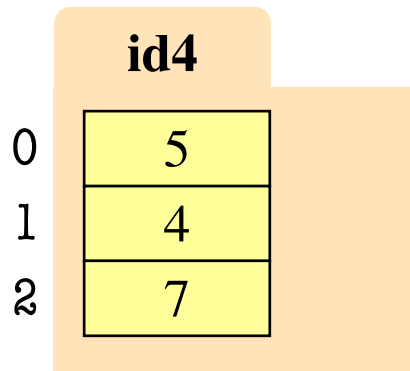
```
1   for x in thelist:
```

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

```
add_one(seq):
```



seq id4



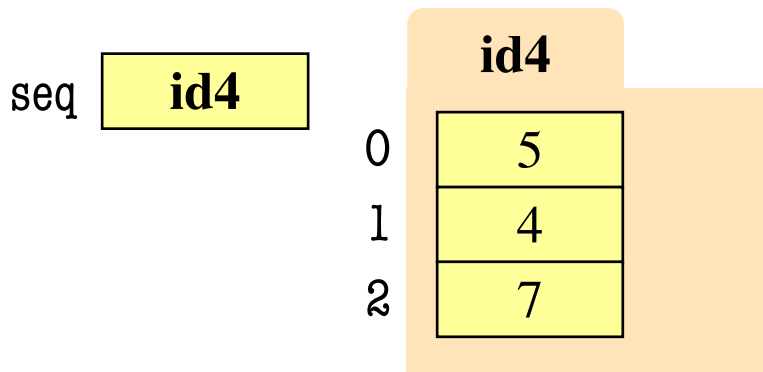
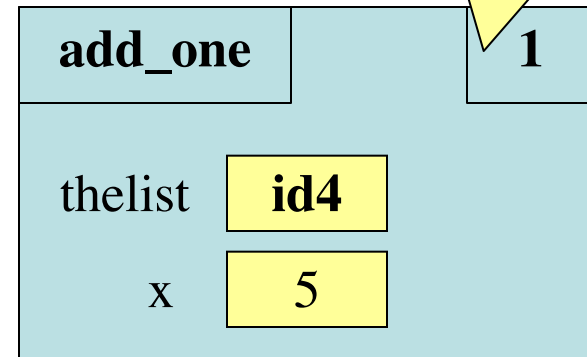
**Next** element stored in x.  
Previous calculation lost.

# For Loops and Call Frames

```
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



# For Loops and Call Frames

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

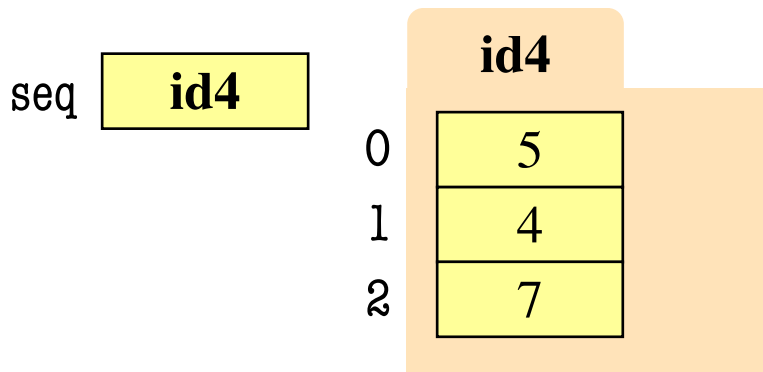
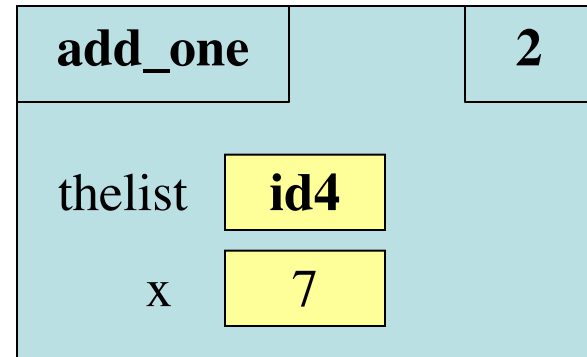
```
    """Adds 1 to every elt
```

```
    Pre: thelist is all numb."""
```

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

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

```
add_one(seq):
```



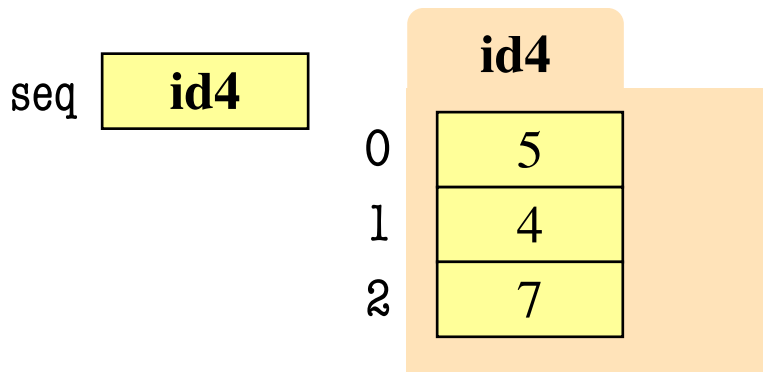
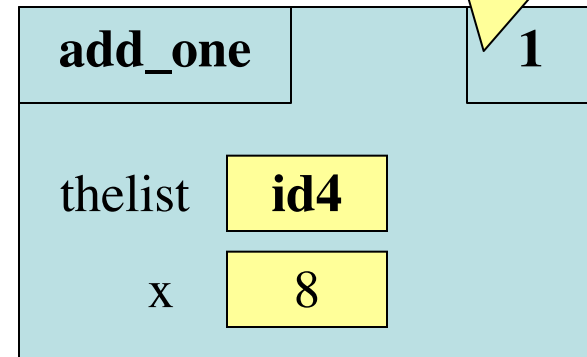
**Next** element stored in x.  
Previous calculation lost.

# For Loops and Call Frames

```
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



# For Loops and Call Frames

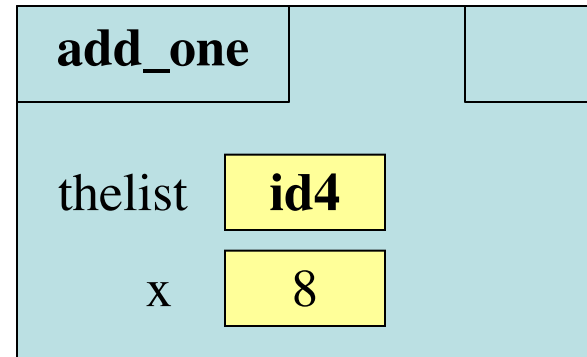
```
def add_one(thelist):
```

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

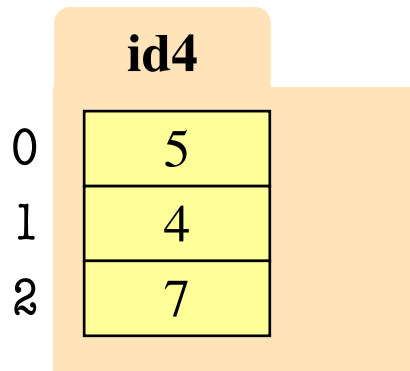
```
1   for x in thelist:
```

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

```
add_one(seq):
```



seq id4



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

# For Loops and Call Frames

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

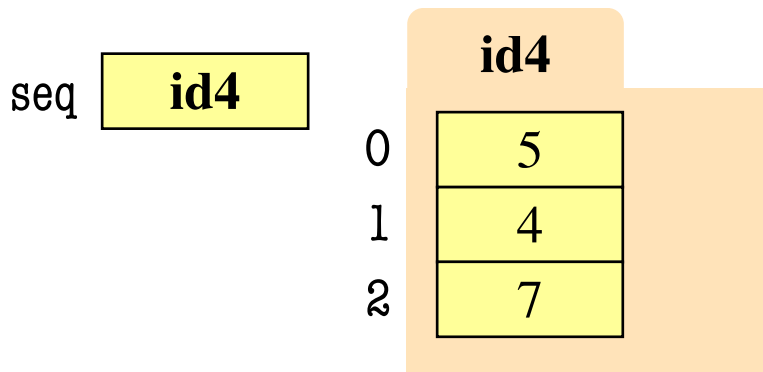
```
    """Adds 1 to every elt  
    Pre: thelist is all numb."""
```

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

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

```
add_one(seq):
```

*ERASE WHOLE FRAME*



No changes  
to folder

# 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)"""
```

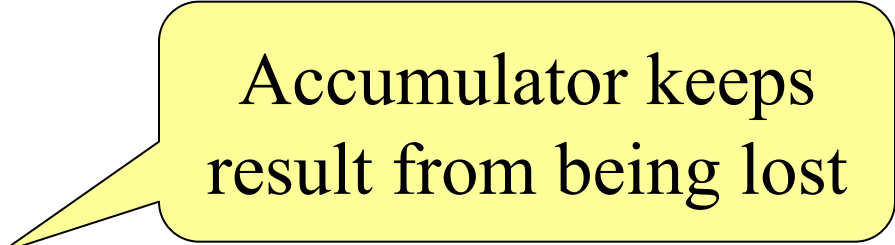
```
    mycopy = [] # accumulator
```

```
    for x in thelist:
```

```
        x = x+1
```

```
        mycopy.append(x) # add to end of accumulator
```

```
    return mycopy
```



Accumulator keeps  
result from being lost



# How Can We Modify A List?

---

- **Never** modify loop var!
- This is an infinite loop:
- Need a second sequence
- How about the *positions*?

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

```
thelist = [5, 2, 7, 1]
```

```
thepos = [0, 1, 2, 3]
```

Try this in Python Tutor  
to see what happens

```
for x in thepos:
```

```
    thelist[x] = thelist[x]+1
```

# How Can We Modify A List?

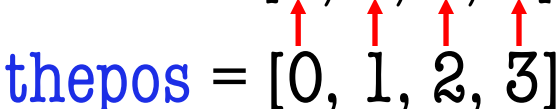
---

- **Never** modify loop var!
- This is an infinite loop:
- Need a second sequence
- How about the *positions*?

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

Try this in Python Tutor  
to see what happens

```
thelist = [5, 2, 7, 1]  
thepos = [0, 1, 2, 3]
```



```
for x in thepos:  
    thelist[x] = thelist[x]+1
```

# 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 to far
- Can convert back & forth
  - `myiter = iter(mylist)`
  - `mylist = list(myiter)`

seq

id1

0  
1  
2

id1

5

4

7

alt

id2

id2

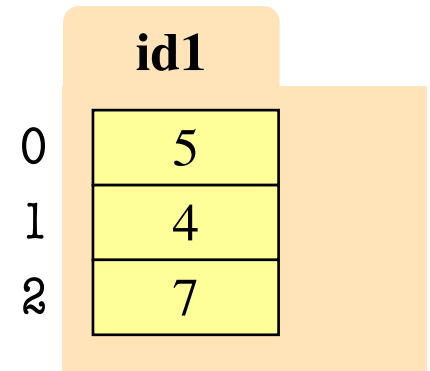
?

Type/Class  
conversion

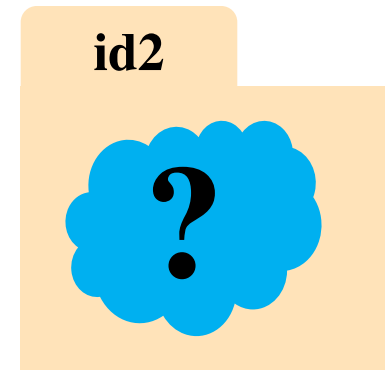
# Iterators and Lists

```
>>> seq = [5, 4, 7]
>>> alt = iter(seq)
>>> next(alt)
5
>>> next(alt)
4
>>> next(alt)
7
>>> next(alt)
Traceback...
```

seq **id1**



alt **id2**



# Iterators and For Loops

```
>>> seq = [5, 4, 7]
```

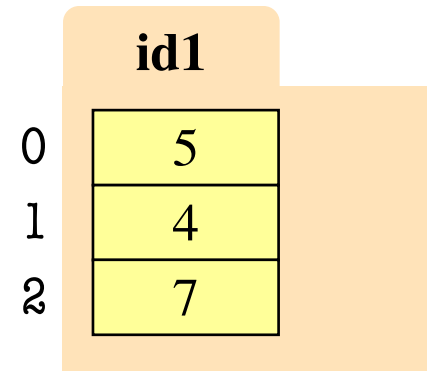
```
>>> alt = iter(seq)
```

```
>>> for x in alt:  
    | print(x)
```

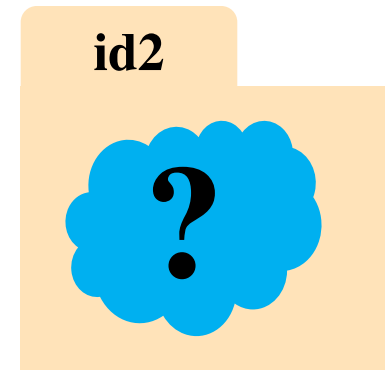
5  
4  
7

Just like looping  
over the list

seq **id1**



alt **id2**



# Iterators and For Loops

```
>>> seq = [5, 4, 7]
```

```
>>> alt = iter(seq)
```

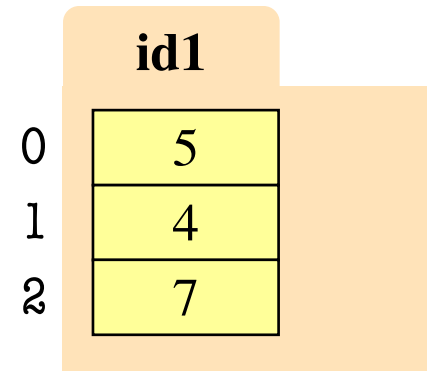
```
>>> for x in alt:  
    | print(x)
```

5  
4  
7

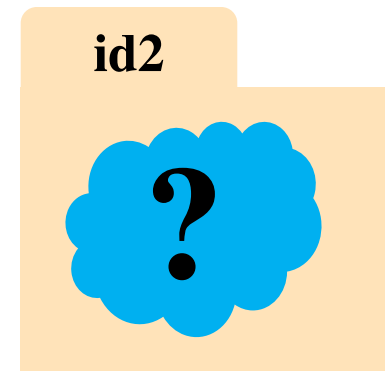
Just like looping  
over the list

But still not safe to  
modify iterator's list

seq **id1**



alt **id2**



# The Range Iterator

---

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

# 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):
```

Iterator of list  
**positions** (safe)

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

```
    # procedure; no return
```

**WORKS!**



# 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 sequence, such as the mean, median, standard deviation
- Send everyone in a Facebook group an appointment time

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

## 2. Perform $n$ trials or get $n$ samples.

- **A4**: draw a triangle six times to make a hexagon
- Run a protein-folding simulation

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

## 3. Do something an unknown number of times

- CUAUV team, vehicle keeps moving until reached its goal

**Cannot do this yet**  
Impossible w/ Python for

