

## Lecture 13

# **For-Loops**

# Announcements for This Lecture

---

## (Optional) Videos

---

- **Videos 16.1-16.7** for **today**
- **Lesson 18** next time
- **Prelim, 10/17 at 7:30 pm**
  - Material up to **10/3**
  - Study guide is posted
  - Fill out the conflict form!
- **Review *this* Wednesday**
  - **5pm in Hollister B14**

## Assignments

---

- A2 is now graded
  - Access it in **Gradescope**
  - Graded out of 50 points
  - **Mean**: 44.5, **Median**: 47
  - **A**: 46 (55%), **B**: 37 (27%)
- A3 due this **Thursday**
  - Time to work on it in lab
  - Will grade over break

# 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)"""
```

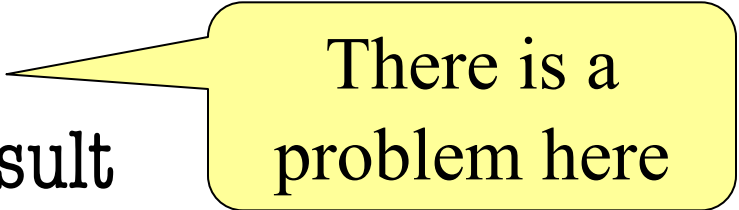
```
    result = 0
```

```
    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**

# The For-Loop

---

```
# Create local var x
```

```
x = seqn[0]
```

```
print(x)
```

```
x = seqn[1]
```

```
print(x)
```

```
...
```

```
x = seqn[len(seqn)-1]
```

```
print(x)
```

Not valid  
Python

```
# Write as a for-loop
```

```
for x in seqn:
```

```
    print(x)
```

---

## Key Concepts

---

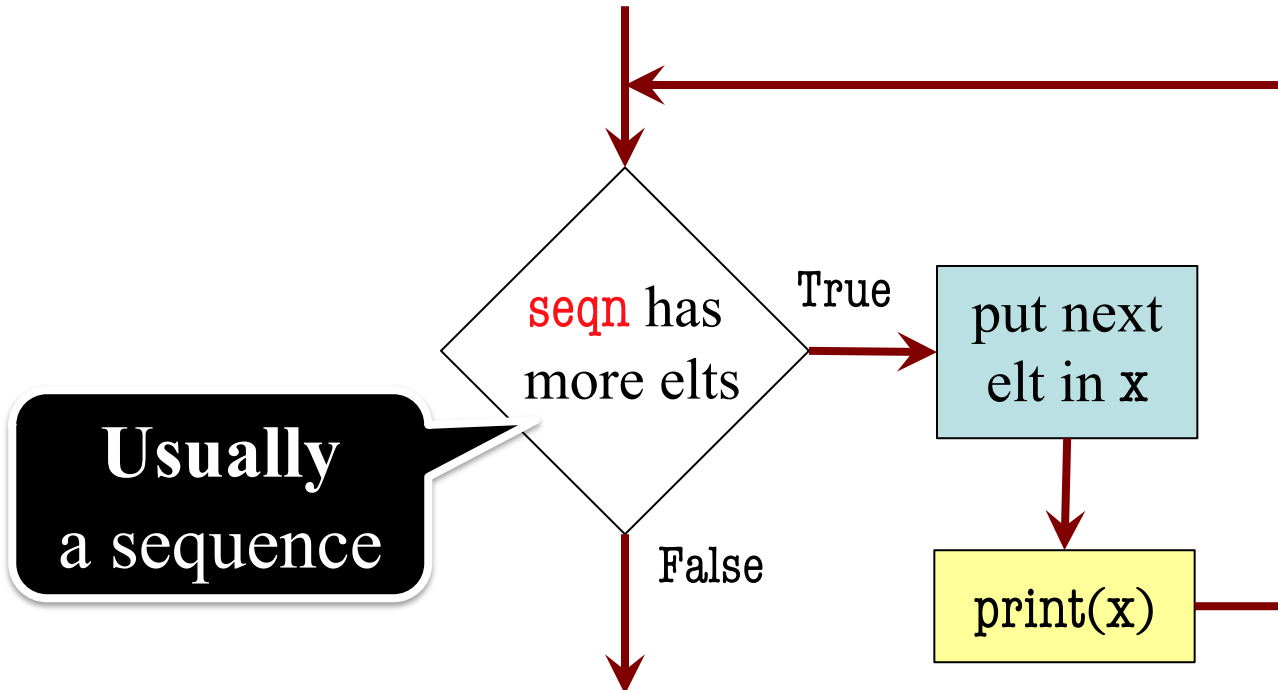
- **iterable:** `seqn`
- **loop variable:** `x`
- **body:** `print(x)`

# Executing a For-Loop

The for-loop:

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

- iterable: **seqn**
- loop variable: **x**
- body: **print(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)"""
```

```
    # 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
```

- **iterable:** 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
```

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

# The Accumulator

---

- In a slides saw the **accumulator**
  - Variable to hold a final (numeric) answer
  - For-loop added to variable at each step
- This is a common *design pattern*
  - Popular way to compute statistics
  - Counting, averaging, etc.
- It is not just limited to numbers
  - Works on **every type that can be added**
  - This means **strings**, **lists** and **tuples**!

# Example: String-Based Accumulator

---

```
def despace(s):  
    """Returns: s but with its spaces removed  
    Precondition: s is a string"""  
    # Create an empty string accumulator  
    # For each character x of s  
        # Check if x is a space  
        # Add it to accumulator if not
```

# Example: String-Based Accumulator

---

```
def despace(s):
```

```
    """Returns: s but with its spaces removed
```

```
    Precondition: s is a string"""
```

```
    result = ''
```

```
    for x in s:
```

```
        if x != ' ':
```

```
            result = result+x
```

**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
```

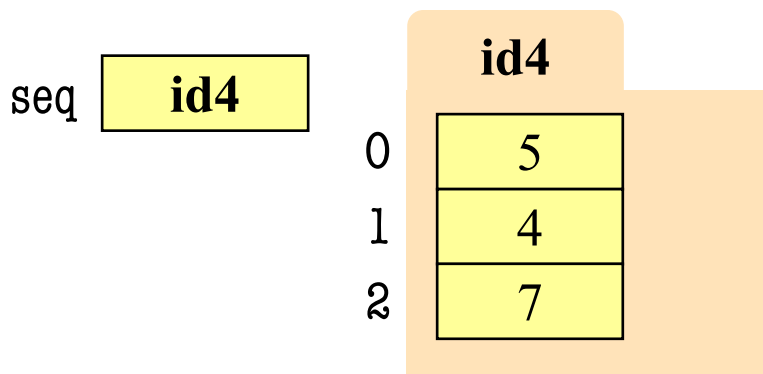
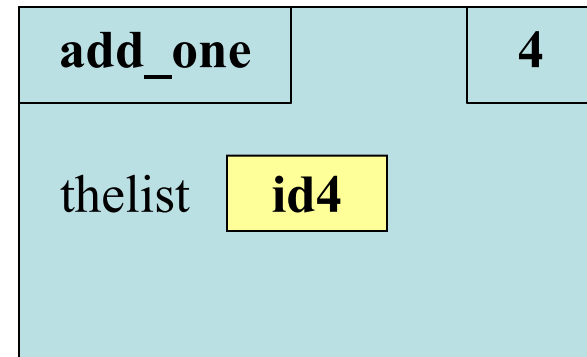
```
    # procedure; no return
```

**DOES NOT WORK!**

# For Loops and Call Frames

```
1. def add_one(thelist):  
2.     """Adds 1 to every elt  
3.     Pre: thelist all nums"""  
4.     for x in thelist:  
5.         x = x+1
```

add\_one(seq):

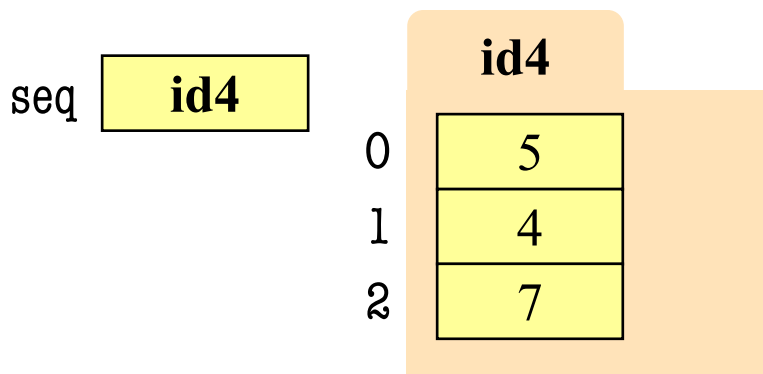
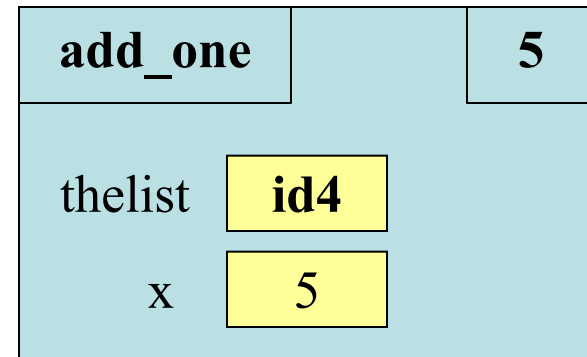




# For Loops and Call Frames

```
1. def add_one(thelist):
2.     """Adds 1 to every elt
3.     Pre: thelist all nums"""
4.     for x in thelist:
5.         x = x+1
```

add\_one(seq):

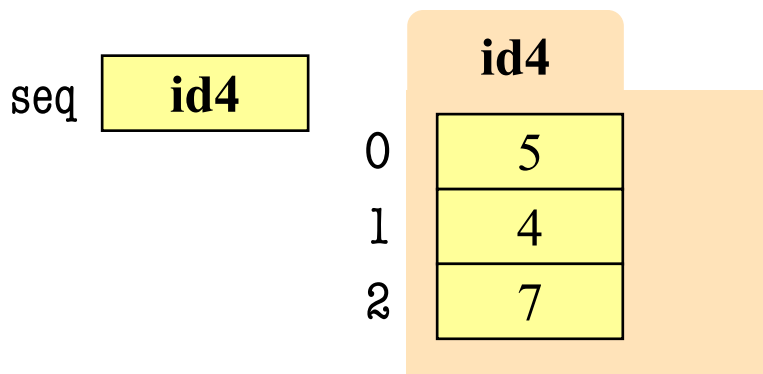
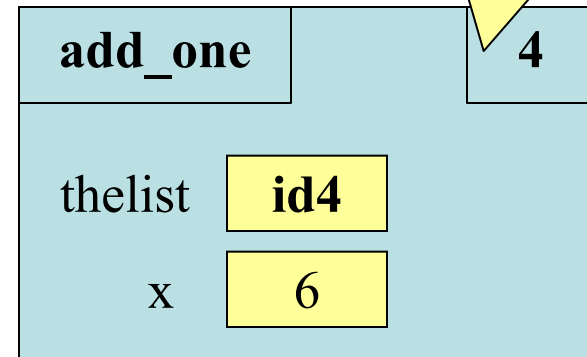


# For Loops and Call Frames

```
1. def add_one(thelist):
2.     """Adds 1 to every elt
3.     Pre: thelist all nums"""
4.     for x in thelist:
5.         x = x+1
```

add\_one(seq):

Loop back  
to line 4

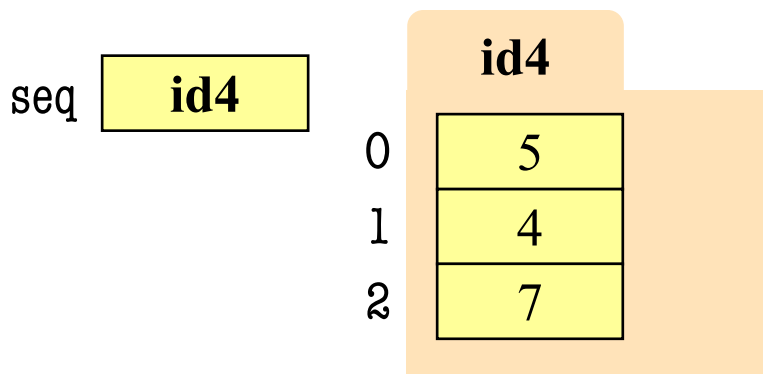
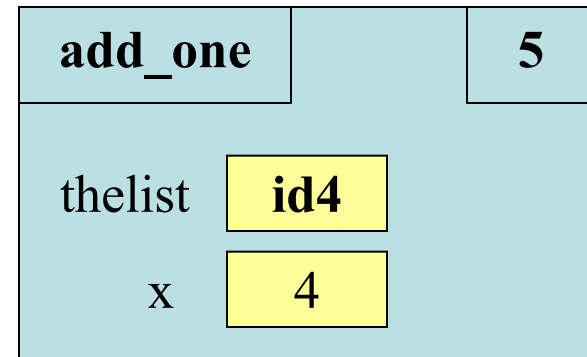


Increments `x` in **frame**  
Does not affect folder

# For Loops and Call Frames

```
1. def add_one(thelist):
2.     """Adds 1 to every elt
3.     Pre: thelist all nums"""
4.     for x in thelist:
5.         x = x+1
```

add\_one(seq):



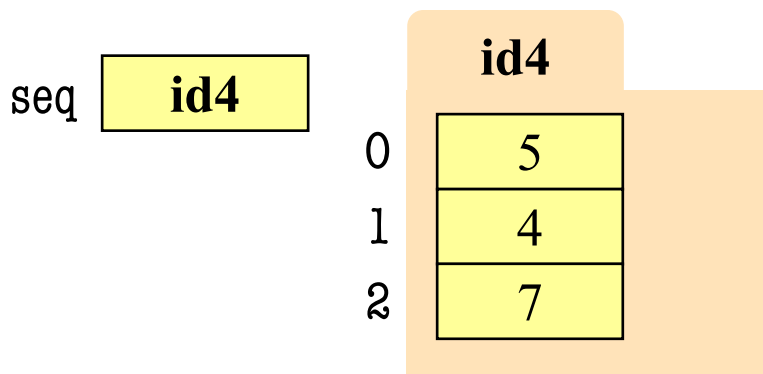
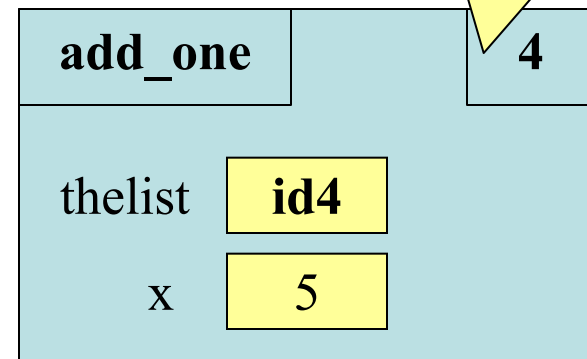
**Next** element stored in x.  
Previous calculation lost.

# For Loops and Call Frames

```
1. def add_one(thelist):  
2.     """Adds 1 to every elt  
3.     Pre: thelist all nums"""  
4.     for x in thelist:  
5.         x = x+1
```

add\_one(seq):

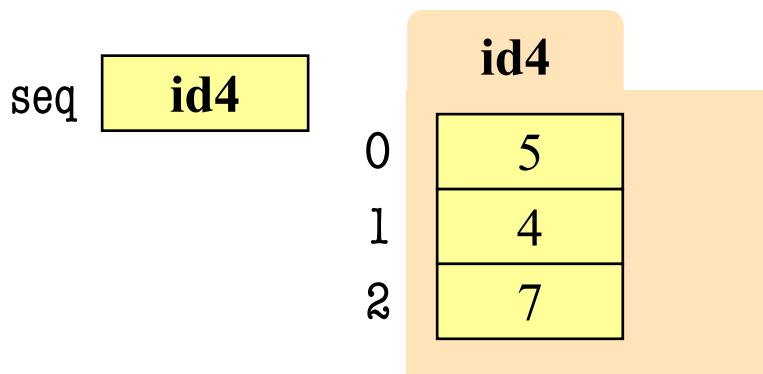
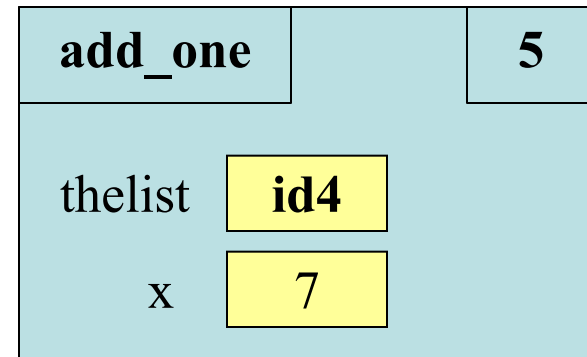
Loop back  
to line 4



# For Loops and Call Frames

```
1. def add_one(thelist):
2.     """Adds 1 to every elt
3.     Pre: thelist all nums"""
4.     for x in thelist:
5.         x = x+1
```

add\_one(seq):



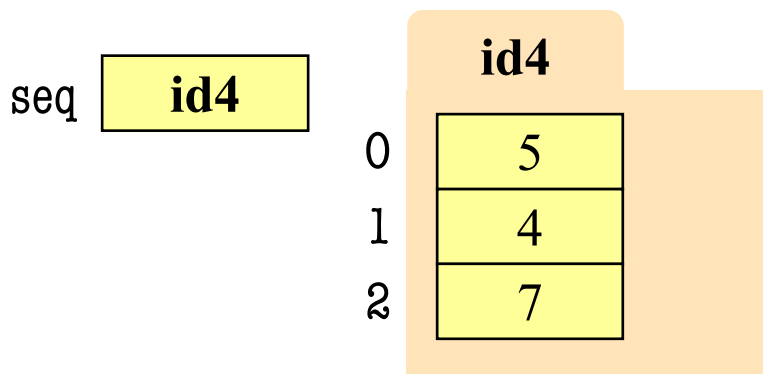
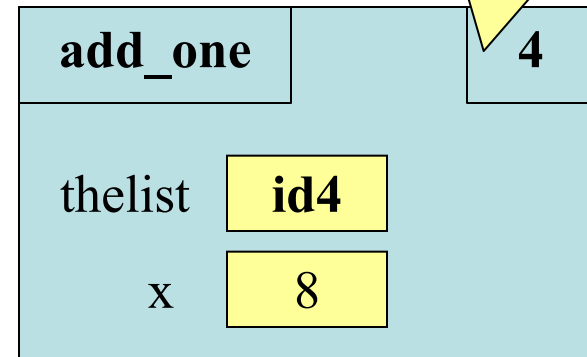
**Next** element stored in x.  
Previous calculation lost.

# For Loops and Call Frames

```
1. def add_one(thelist):
2.     """Adds 1 to every elt
3.     Pre: thelist all nums"""
4.     for x in thelist:
5.         x = x+1
```

add\_one(seq):

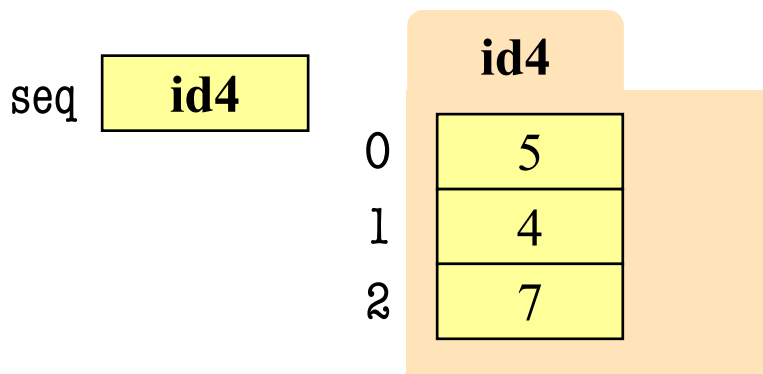
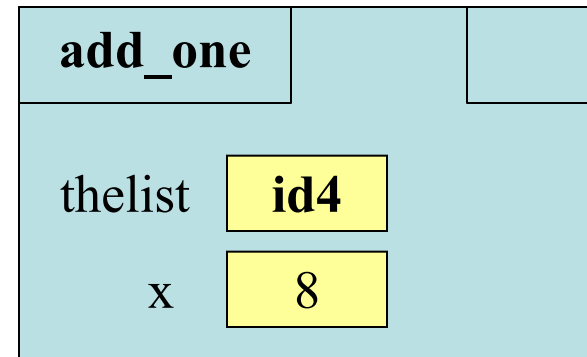
Loop back  
to line 4



# For Loops and Call Frames

```
1. def add_one(thelist):  
2.     """Adds 1 to every elt  
3.     Pre: thelist all nums"""  
4.     for x in thelist:  
5.         x = x+1
```

add\_one(seq):



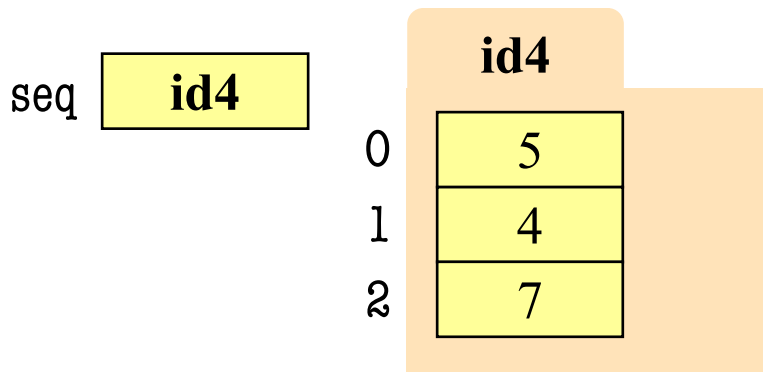
Loop is **completed**.  
Nothing new put in x.

# For Loops and Call Frames

```
1. def add_one(thelist):  
2.     """Adds 1 to every elt  
3.     Pre: thelist all nums"""  
4.     for x in thelist:  
5.         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)"""
```

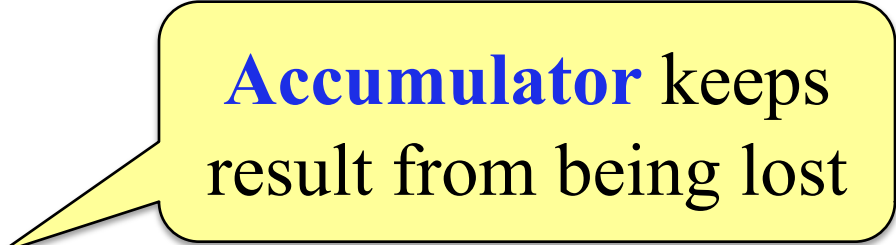
```
    mycopy = [] # accumulator
```

```
    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 iterable!
- 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 in Python Tutor  
to see what happens

```
for x in thepos:
```

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

# How Can We Modify A List?

---

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

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

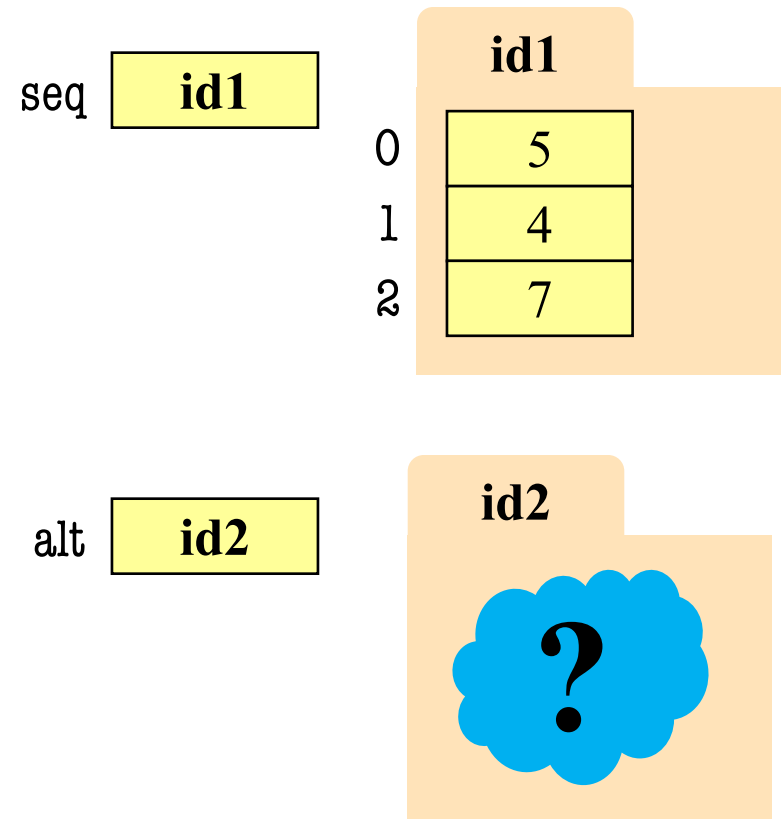
```
thelist = [5, 2, 7, 1]  
↑ ↑ ↑ ↑  
thepos = [0, 1, 2, 3]
```

Try in Python Tutor  
to see what happens

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

# This is the Motivation for Iterables

- **Iterables** are objects
  - Contain data like a list
  - **But cannot slice them**
- Have list-like properties
  - Can use them in a for-loop
  - Can convert them to lists
  - `mylist = list(myiterable)`
- **Example:** Files
  - Use `open()` to create object
  - Makes iterable for reading



# Iterables, Lists, and For-Loops

```
>>> file = open('sample.txt')
```

```
>>> list(file)
```

```
['This is line 1\n',  
 'This is line 2\n']
```

```
>>> file = open('sample.txt')
```

```
>>> for line in file:
```

```
...     print(line)
```

```
This is line one
```

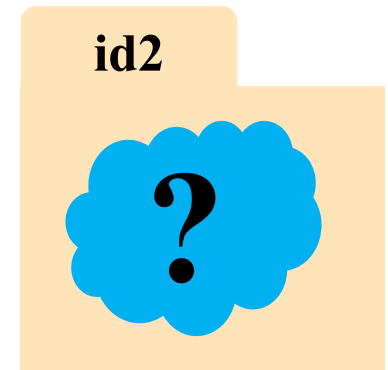
```
This is line two
```

print adds \n  
in *addition* to  
one from file

seq **id1**

id1	
0	5
1	4
2	7

alt **id2**



# The Range Iterable

---

- `range(x)`
  - Creates an iterable
  - 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

