



Lecture 11: **Iteration and For-Loops**

(Sections 4.2 and 10.3)

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Important concept in computing:

Doing things repeatedly

1. Perform n trials or get n samples.

- Run a protein-folding simulation for 10^6 time steps
- Next 50 ticket purchases entered in random draw for upgrade

2. Process each item in a sequence

Repeat a known (definite)
number of times

- Compute aggregate statistics (e.g., mean, median) on scores
- Send everyone in a Facebook group an appointment time

3. Do something an unknown number of times

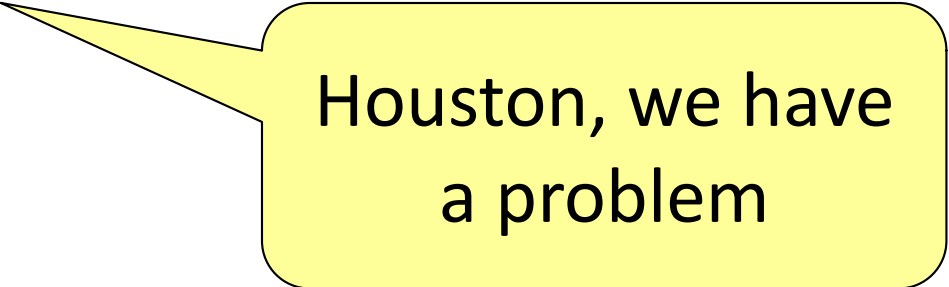
- CUAUV team, vehicle keeps moving until reached its goal

Repeat until something happens—
repeat an indefinite number of times



1st Attempt: Summing the Elements of a List

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



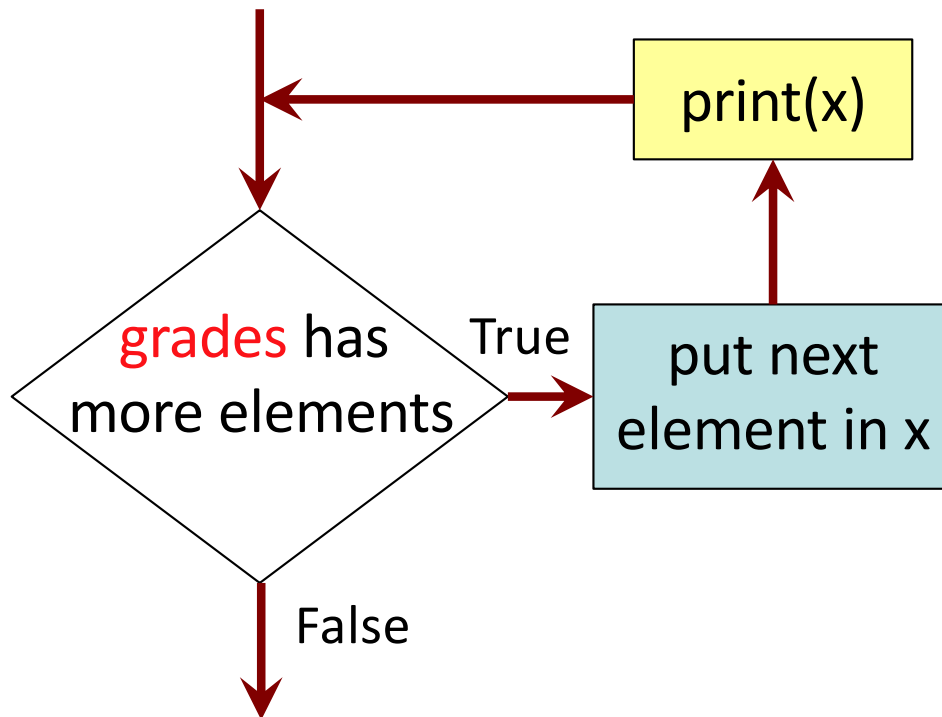
Houston, we have
a problem

Working with Sequences

- Sequences are potentially **unbounded**
 - Number of elements 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 there are millions of elements?
- We need a new approach

For Loops: Processing Sequences

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



- **loop sequence:** grades
- **loop variable:** x
- **loop body:** print(x)

To execute the for-loop:

- 1) Check if there is a “next” element of **loop sequence**
- 2) If so:
 - assign next sequence element to **loop variable**
 - Execute all of **the body**
 - Go back to 1)
- 3) If not, terminate execution

Solution: Summing the Elements of a List

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

For Loops and Conditionals

```
def num_zeroes(the_list):  
    """Returns: the number of zeroes in the_list  
    Precondition: the_list is a list"""  
  
    count = 0                # Create var. to keep track of 0's  
    for x in the_list:       # for each element in the list..  
        if x == 0:           # check if it is equal to 0  
            count = count + 1 # add 1 if it is  
    return count             # Return the variable/counter
```

For Loop with labels

```
def num_zeroes(the_list):  
    """Returns: the number of zeroes in the_list  
    Precondition: the_list is a list"""
```

```
    count = 0
```

```
    for x in the_list:
```

```
        if x == 0:
```

```
            count = count + 1
```

```
    return count
```

Accumulator variable

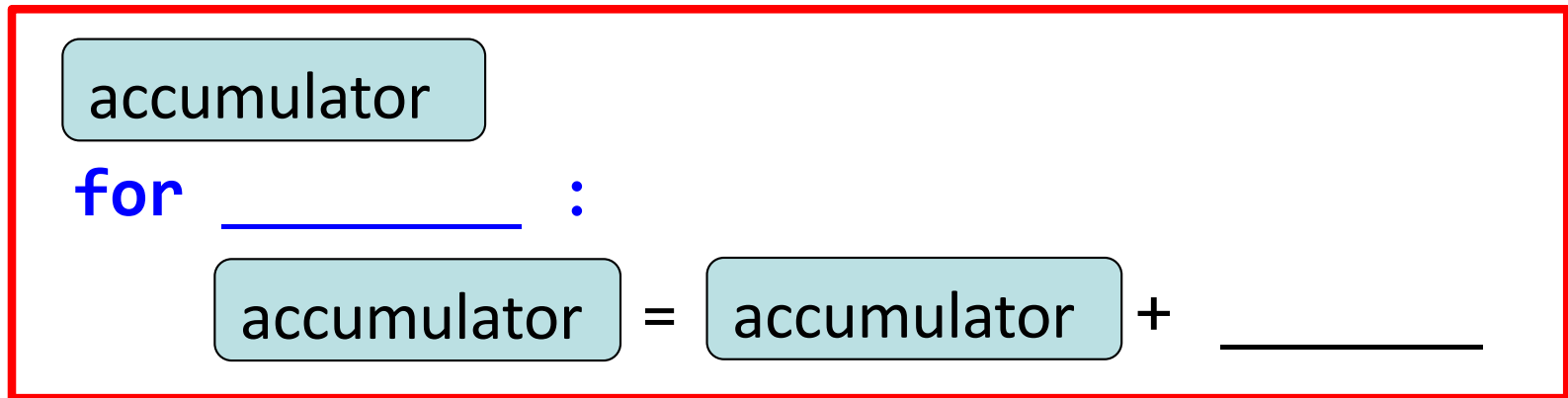
Loop sequence

Loop variable

Loop body

Accumulator

- A variable to hold a final answer
- for-loop adds to the variable at each step
- The final answer is accumulated, i.e., built up, one step at a time. A common design *pattern*:



- Accumulator does not need to be a number. E.g., can be a string to be built-up

Exercise

```
def ave_positives(my_list):  
    """Returns: avg (float) of positive values in my_list  
    my_list: a list of #s with at least 1 positive value  
    """
```

- Be goal oriented → *can work backwards*
- *Name a variable* for any value that you need but don't have yet
- Break down a problem!
 - ... *break into parts*
 - ... *solve simpler version first*
- Remember loop/accumulation pattern

What if we aren't dealing with a list?

So far we've been building for-loops around elements of a list.

What if we just want to do something some number of times?

range to the rescue!

range: a handy counting function!

`range(x)`

generates 0,1,...,x-1

```
>>> print(range(6))  
range(0, 6)
```

Important: range does not return a list

can to convert `range`'s return value into a list

`range(a,b)`

→ a,...,b-1

Arguments must
be int expressions

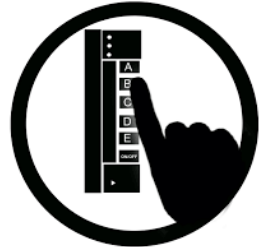
```
>>> first_six = list(range(6))  
>>> print(first_six)  
[0, 1, 2, 3, 4, 5]
```

`range(a,b,s)`

→ a,a+s,a+2s,...,b-1

```
>>> second_six = list(range(6,13))  
>>> print(second_six)  
[6, 7, 8, 9, 10, 11, 12]
```

What gets printed? (Q)



```
t= 0
for k in range(5, 1, -1):
    t = t + 1
print(t)
```

A: 0
B: 2
C: 3
D: 4
E: 5

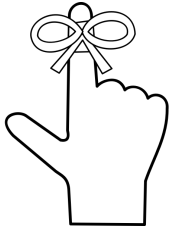
Modifying the Contents of a List

```
def add_bonus(grades):  
    """Adds 1 to every element in a list of grades  
    (either floats or ints)"""  
    size = len(grades)  
    for k in range(size):  
        grades[k] = grades[k]+1
```

*If you need to
modify the list, you
need to use range to
get the indices.*

```
lab_scores = [8,9,10,5,9,10]  
print("Initial grades are: "+str(lab_scores))  
add_bonus(lab_scores)  
print("With bonus, grades are: "+str(lab_scores))
```

*Watch this in the
python tutor!*



For-Loop Tip #1

Remember to modify the list, not just the loop variable.

For-Loop Tip #1 (Q)



Modifying the loop variable (here: x).

```
def add_one(the_list):  
    """Adds 1 to every element in the list  
    Precondition: the_list is a list of all numbers  
    (either floats or ints)"""  
    for x in the_list:  
        x = x+1
```

What gets printed?

```
a = [5, 4, 7]  
add_one(a)  
print(a)
```

A: [5, 4, 7]

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

C: [6, 5, 8]

D: Error

E: I don't know

Modifying the Loop Variable (*oops!*) (1)

```
def add_one(the_list):
```

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

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

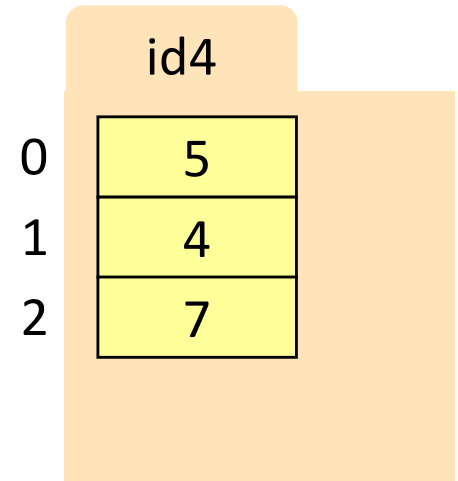
```
1  for x in the_list:
```

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

Global Space

grades id4

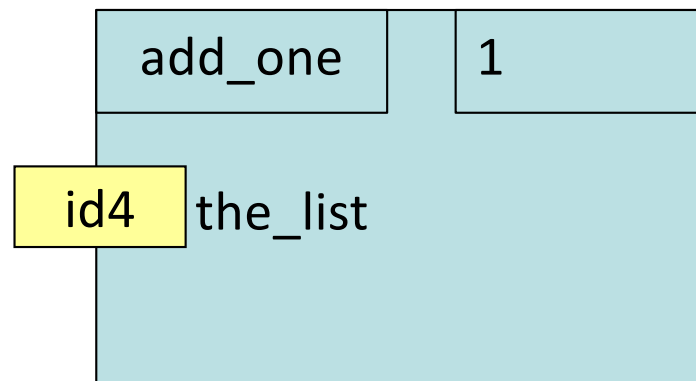
Heap Space



```
grades = [5,4,7]
```

```
add_one(grades)
```

Call Frame



Modifying the Loop Variable (*oops!*) (2)

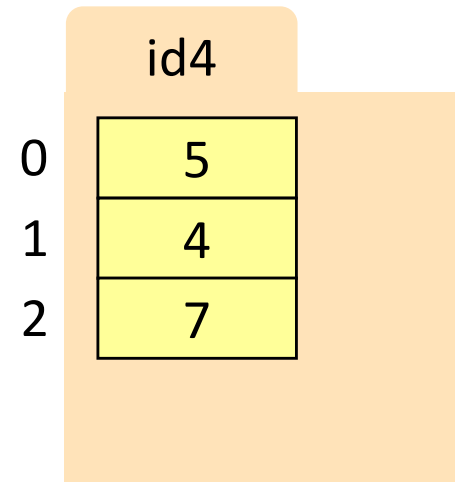
```
def add_one(the_list):  
    """Adds 1 to every elt  
    Pre: the_list is all numb."""  
1  for x in the_list:  
2  → x = x+1
```

```
grades = [5,4,7]  
add_one(grades)
```

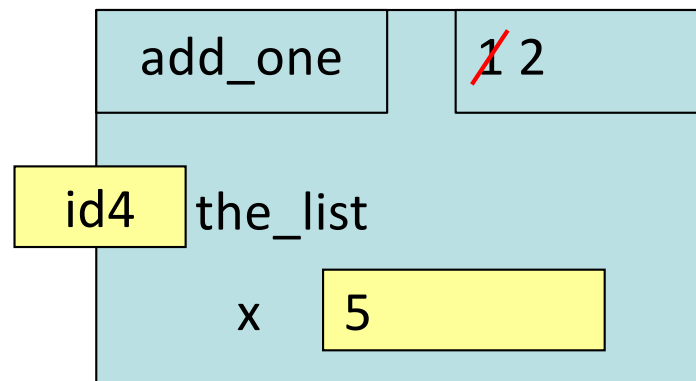
Global Space

grades id4

Heap Space



Call Frame



Modifying the Loop Variable (*oops!*) (3)

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

Global Space

Heap Space

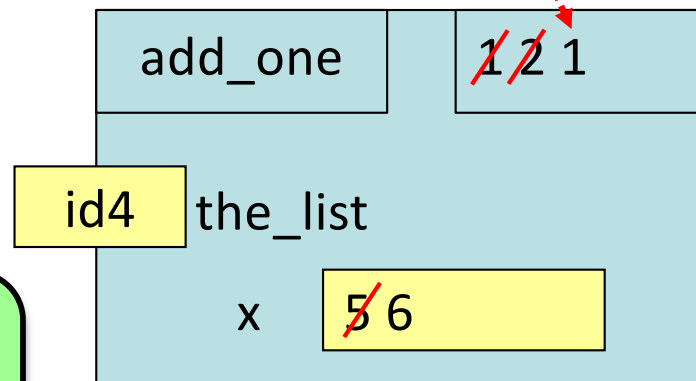
grades id4

	id4
0	5
1	4
2	7

```
grades = [5,4,7]  
add_one(grades)
```

Loop back
to line 1

Call Frame



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

Modifying the Loop Variable (*oops!*) (4)

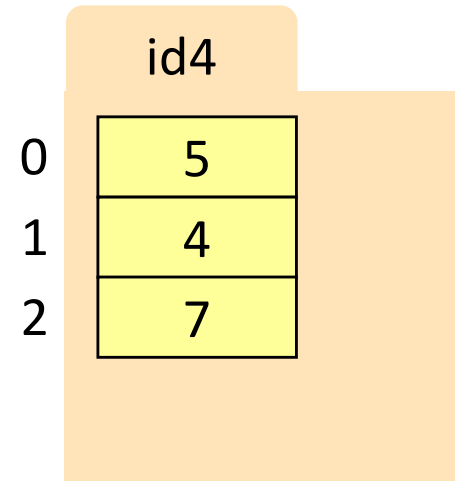
```
def add_one(the_list):  
    """Adds 1 to every elt  
    Pre: the_list is all numb."""  
1  for x in the_list:  
2  → x = x+1
```

```
grades = [5,4,7]  
add_one(grades)
```

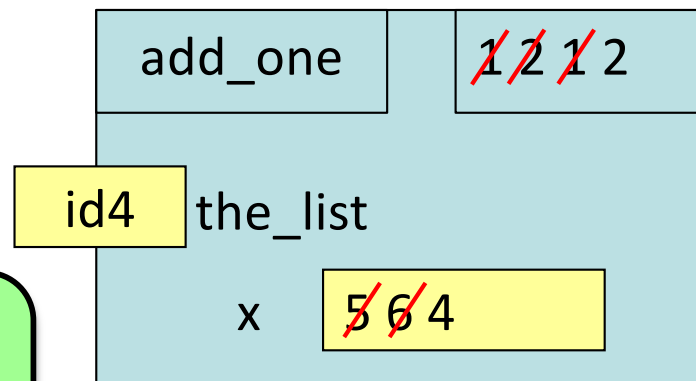
Global Space

grades id4

Heap Space



Call Frame



Next element stored in x.
Previous calculation lost.

Modifying the Loop Variable (*oops!*) (5)

```
def add_one(the_list):
```

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

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

```
1  for x in the_list:
```

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

```
grades = [5,4,7]
```

```
add_one(grades)
```

Global Space

grades id4

Heap Space

id4

0

5

1

4

2

7

Loop back
to line 1

Call Frame

add_one

~~1~~ ~~2~~ ~~1~~ ~~2~~ ~~1~~

id4

the_list

x

~~5~~ ~~6~~ ~~4~~ ~~5~~

Modifying the Loop Variable (*oops!*) (6)

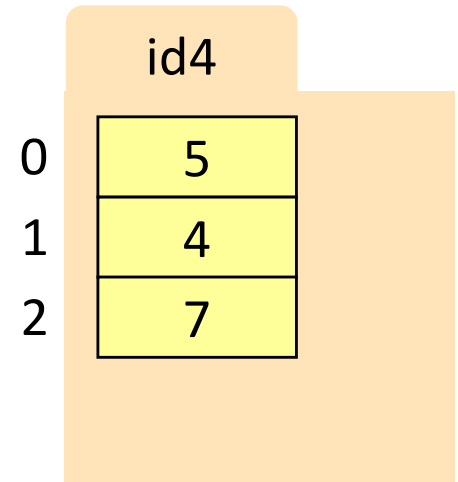
```
def add_one(the_list):  
    """Adds 1 to every elt  
    Pre: the_list is all numb."""  
1  for x in the_list:  
2  → x = x+1
```

```
grades = [5,4,7]  
add_one(grades)
```

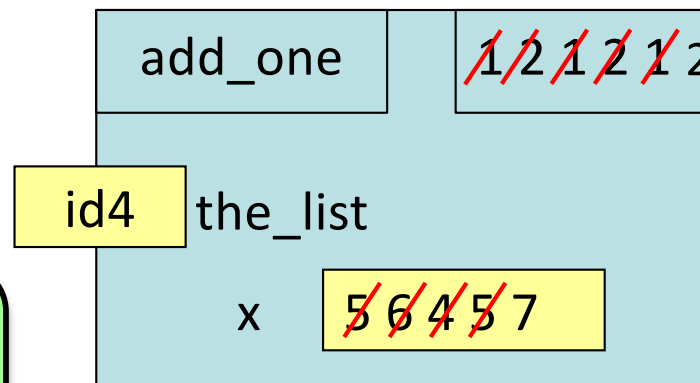
Global Space

grades id4

Heap Space



Call Frame



Next element stored in x.
Previous calculation lost.

Modifying the Loop Variable (*oops!*) (7)

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

```
grades = [5,4,7]  
add_one(grades)
```

Global Space

grades id4

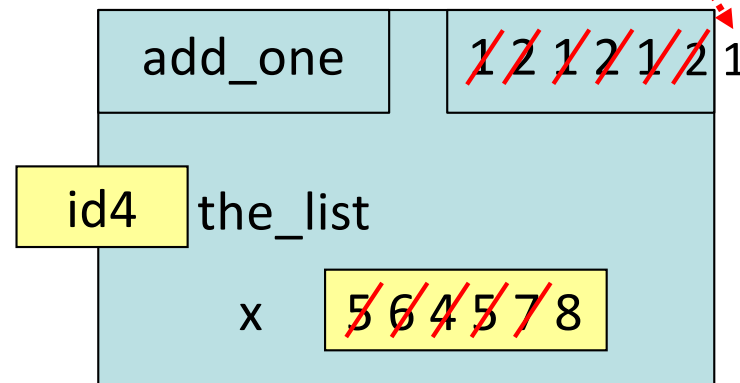
Heap Space

id4

0	5
1	4
2	7

Loop back
to line 1

Call Frame

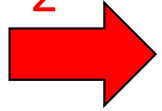


Modifying the Loop Variable (*oops!*) (8)

```
def add_one(the_list):  
    """Adds 1 to every elt  
    Pre: the_list is all numb."""
```

```
1  for x in the_list:
```

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

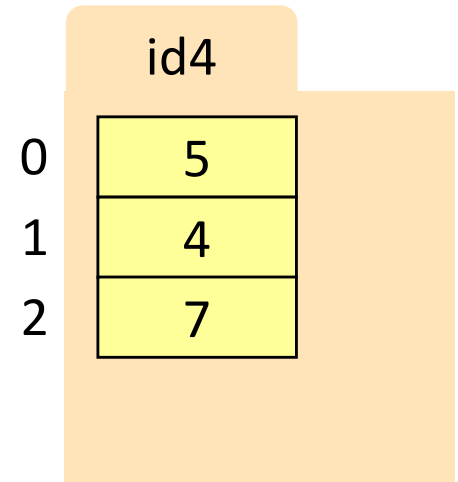


```
grades = [5,4,7]  
add_one(grades)
```

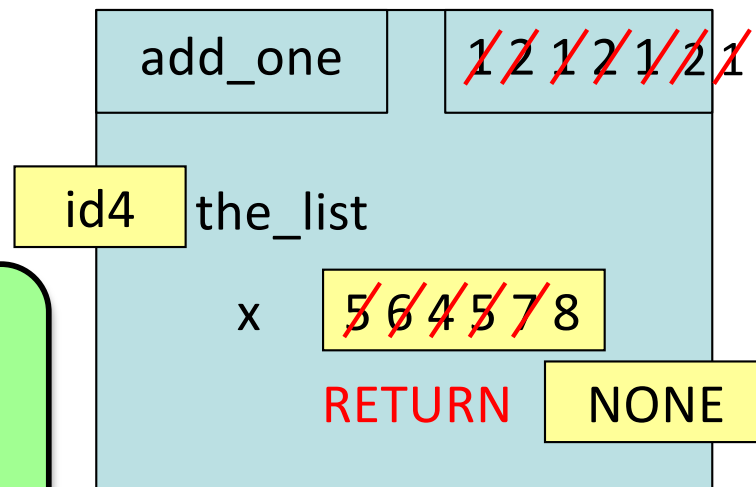
Global Space

grades id4

Heap Space



Call Frame



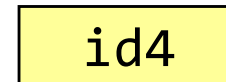
Loop is **completed**.
Nothing new put in x.

Modifying the Loop Variable (*oops!*) (9)

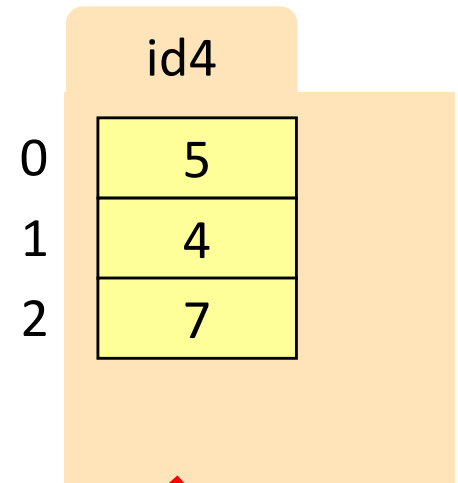
```
def add_one(the_list):  
    """Adds 1 to every elt  
    Pre: the_list is all numb."""  
1  for x in the_list:  
2      x = x+1
```

Global Space

grades



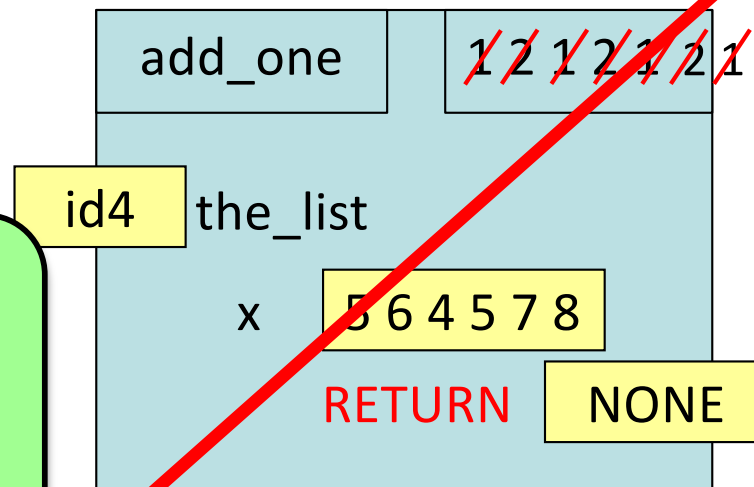
Heap Space



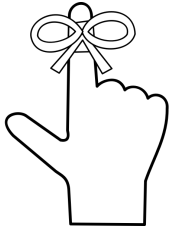
grades = [5,4,7]

add_one(grades)

Call Frame



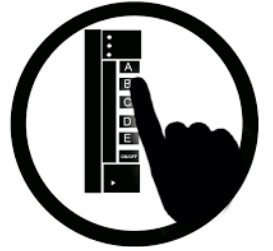
No lasting changes.
What did we
accomplish? ☹️



For-Loop Tip #2

Don't modify the loop sequence as you walk through it.

For-Loop Tip #2 (Q)



Modifying the loop sequence as you walk through it.

What gets printed?

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

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

Iterating through a dictionary

```
students = {'ec1': 'Ezra', 'ec2': 'Ezra',  
            'tm55': 'Toni'}
```

```
# loop variable iterates through each key  
for netID in students:  
    print(netID + ":" + students[netID])
```

prints to the screen:

```
ec1:Ezra  
ec2:Ezra  
tm55:Toni
```