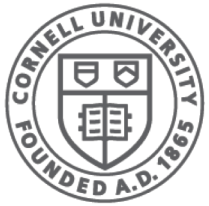
http://www.cs.cornell.edu/courses/cs1110/2019sp

Lecture 12: Iteration and For-Loops

(Sections 4.2 and 10.3)

CS 1110

Introduction to Computing Using Python



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

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

Problem: 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)"""
```

Approach: 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)"""
    # Create a variable to hold result (start at 0)
    # Add each list element to variable
    # Return the variable
```

How will we do this?

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
```

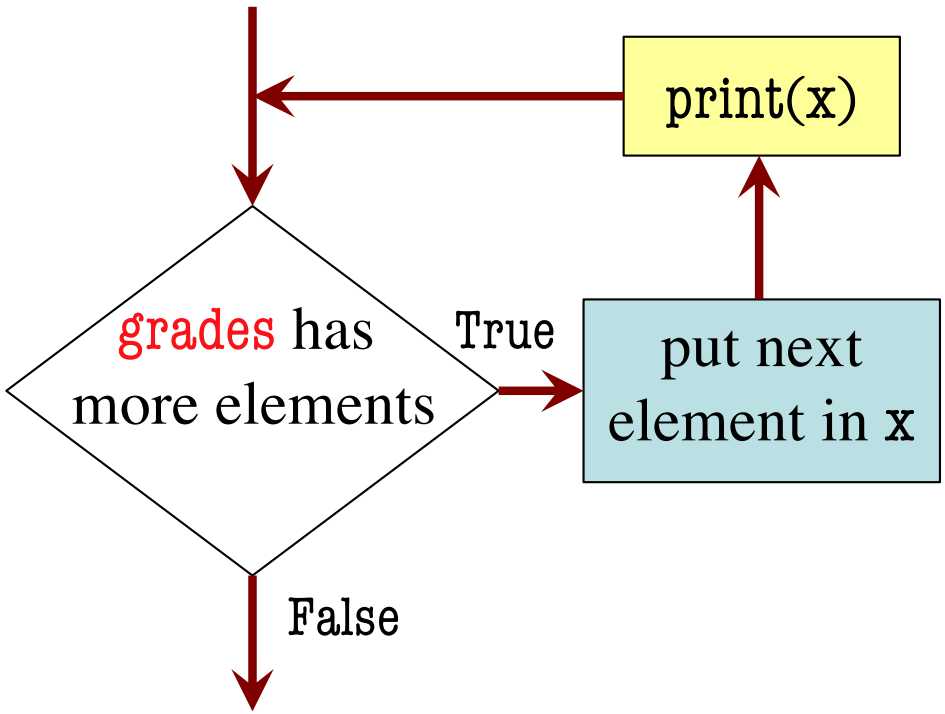
Houston, we have a problem

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
- **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 Line 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
```

Accumulator variable

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

What gets printed? (Q1)

my_list = [1]

```
s = 0
for x in my_list:
    s = s + x
print(s)
```

same code

my_list = [1,7,2]

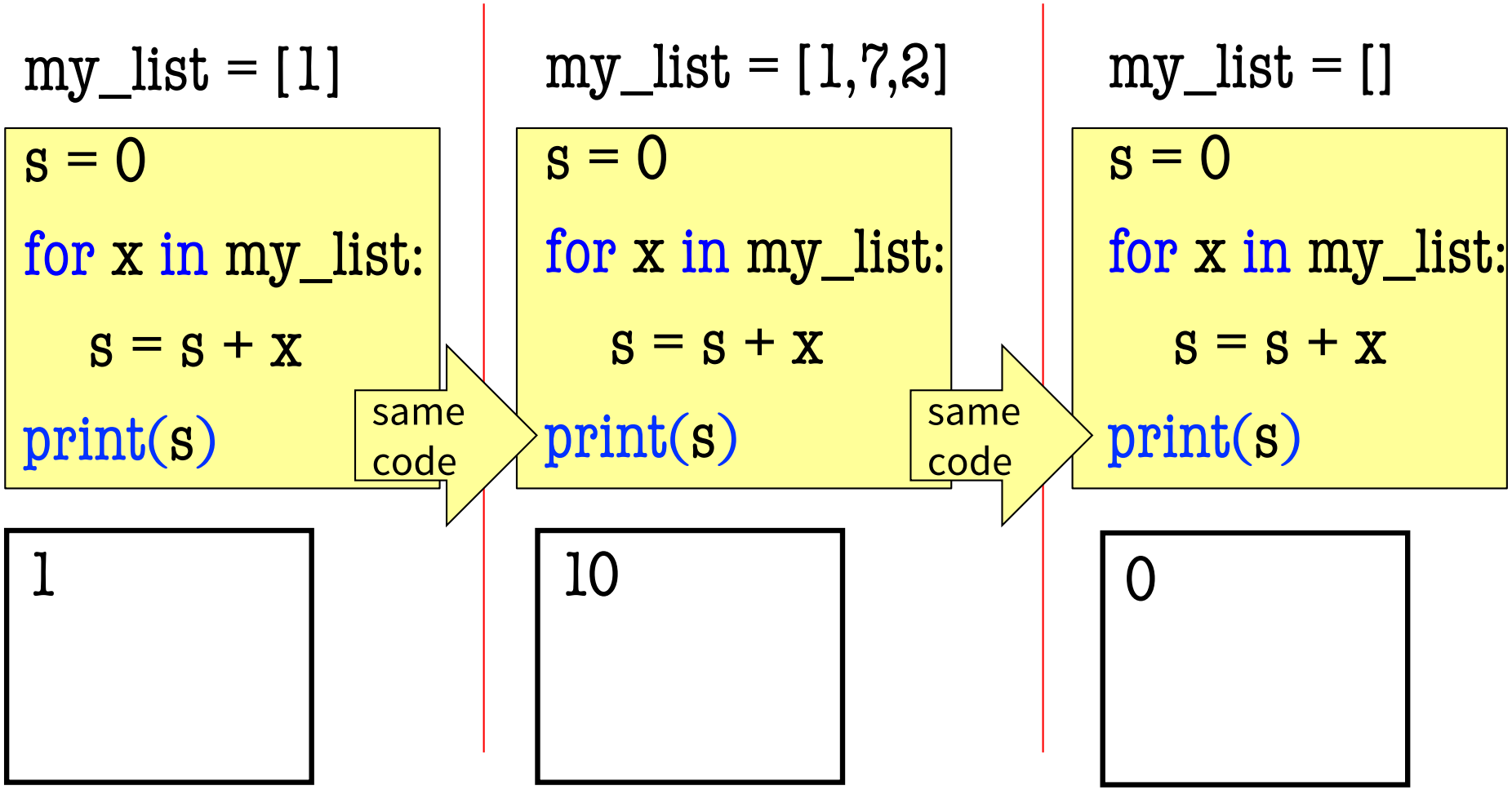
```
s = 0
for x in my_list:
    s = s + x
print(s)
```

same code

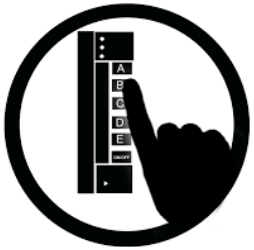
my_list = []

```
s = 0
for x in my_list:
    s = s + x
print(s)
```

What gets printed? (A1)



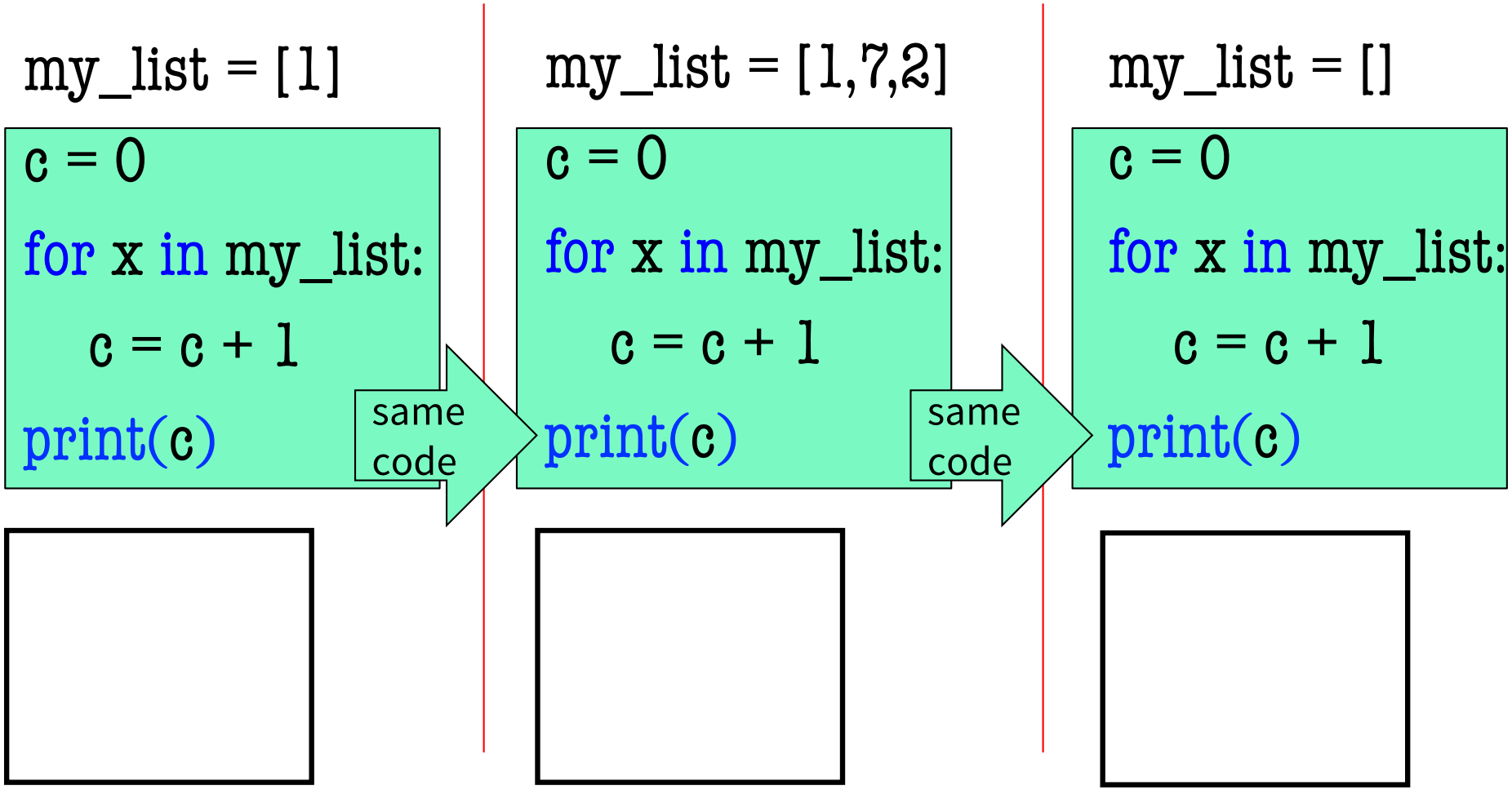
What does this loop do?



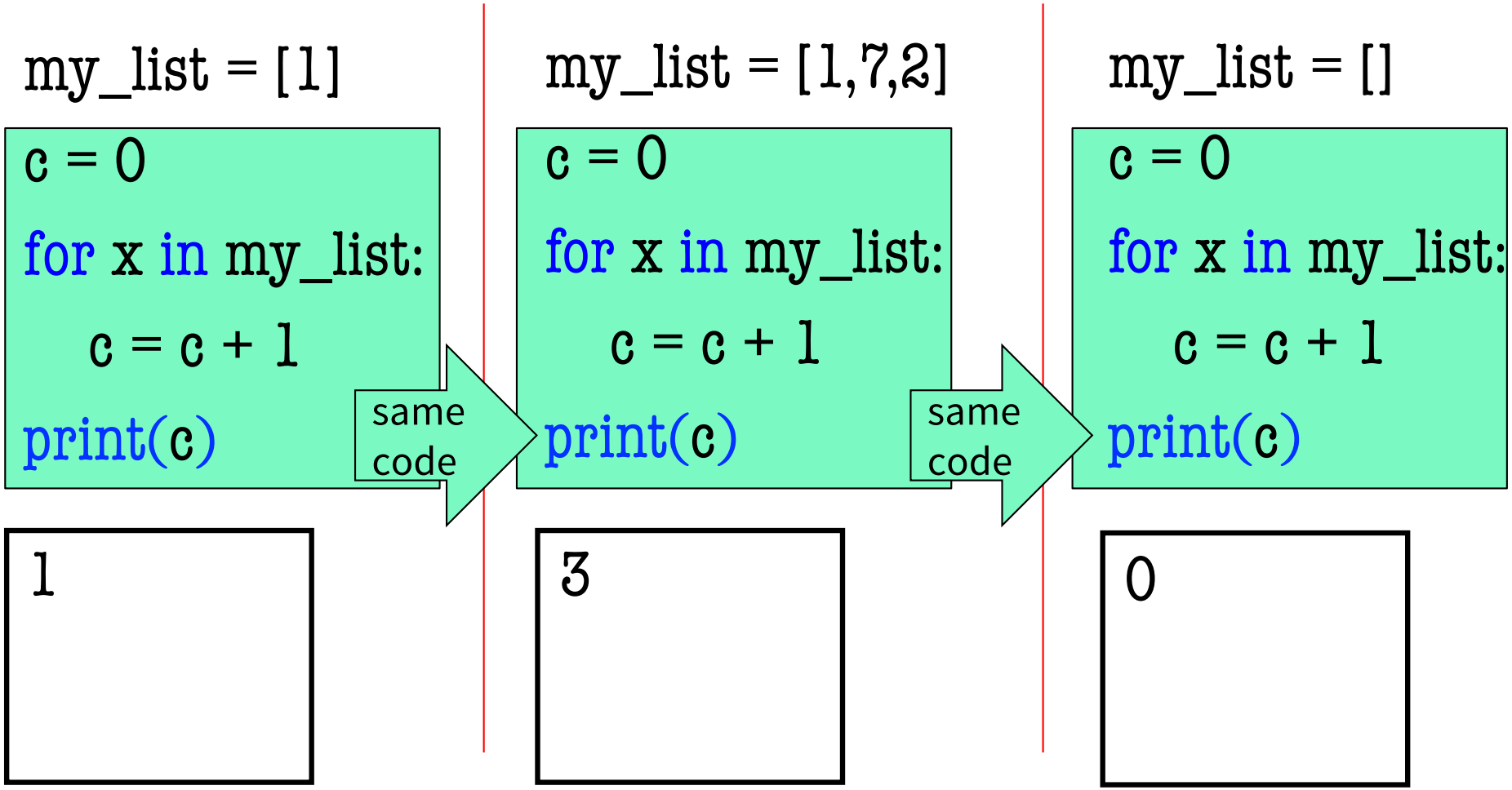
```
my_list = [1]
s = 0
for x in my_list:
    s = s + x
print(s)
```

A: it sums the elements in my_list
B: it prints the elements in my_list
C: it counts the elements in my_list
D: it adds one to the elements in my_list
E: none of the above

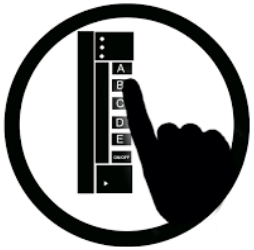
What gets printed? (Q1)



What gets printed? (A1)



What does this loop do?



```
my_list = [1]
c = 0
for x in my_list:
    c = c + 1
print(c)
```

A: it sums the elements in my_list
B: it prints the elements in my_list
C: it counts the elements in my_list
D: it adds one to the elements in my_list
E: none of the above

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

Body

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)

returns 0,1,…,x-1

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

Important: range does not return a list

→ need to convert ranges' return value into a list

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

range(a,b)

returns a,…,b-1

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

range in a for-loop, v1

```
for num in range(5):
    print(str(num))
print("Once I caught a fish alive.")
```

```
0
1
2
3
4
Once I caught a fish alive.
```

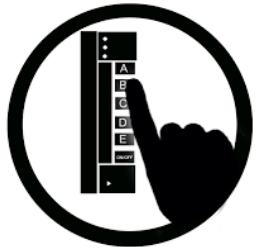
range in a for-loop, v2

```
for num in range(1,6):
    print(str(num))
print("Once I caught a fish alive.")

for num in range(6,11):
    print(str(num))
print("Then I let him go again.")
```

```
1
2
3
4
5
Once I caught a fish alive.
6
7
8
9
10
Then I let him go again.
```

What gets printed?



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

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

Modifying the Contents of a List

```
def inflate_grades(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))
inflate_grades(lab_scores)
print("Inflated grades are: "+str(lab_scores))
```

Watch this in the python tutor!

Common For-Loop Mistakes (1)

Mistake #1: Modifying the loop variable instead of the list itself.

For-Loop Mistake #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
```

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

For-Loop Mistake #1 (A)

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
```

Actually it does not do this!

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

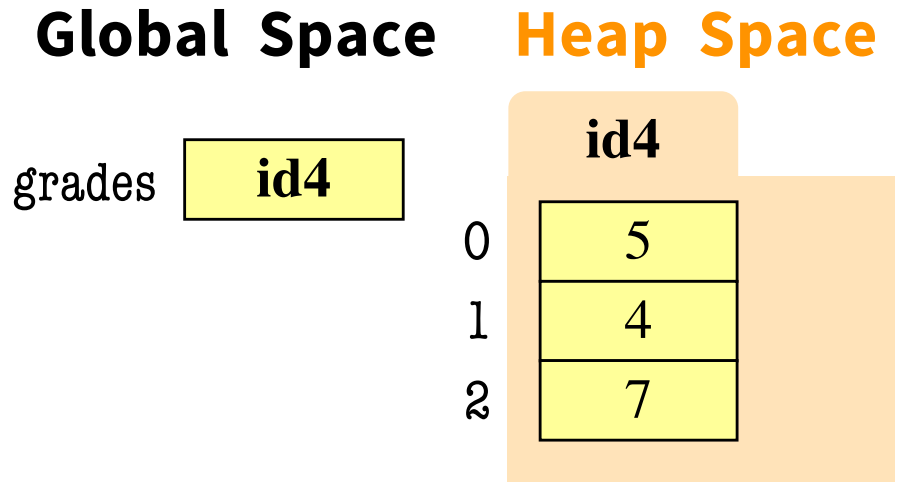
What gets printed?

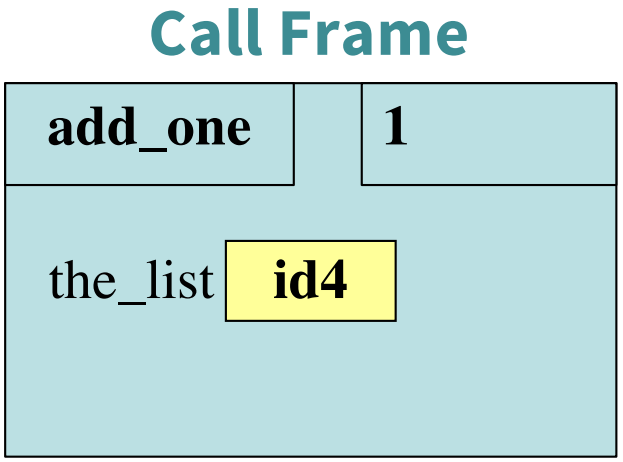
A: [5, 4, 7] **CORRECT**
B: [5, 4, 7, 5, 4, 7]
C: [6, 5, 8]
D: **Error**
E: I don't know

Modifying the Loop Variable (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
```

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

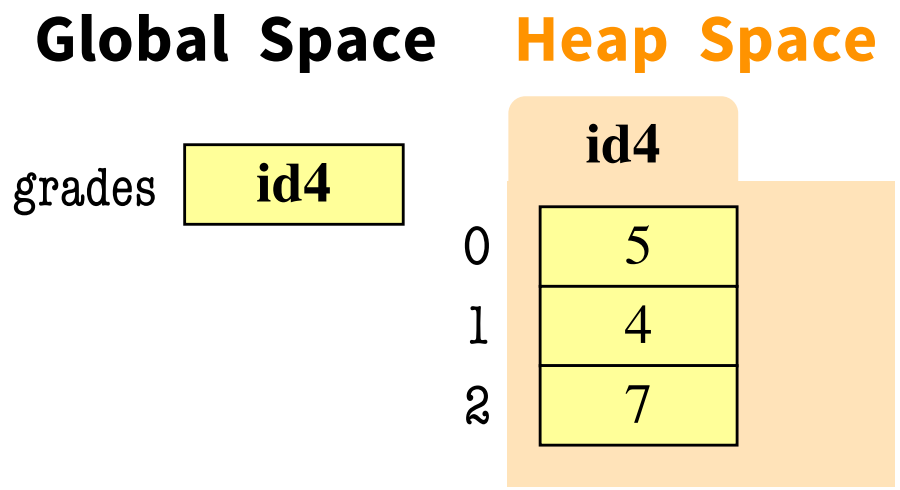


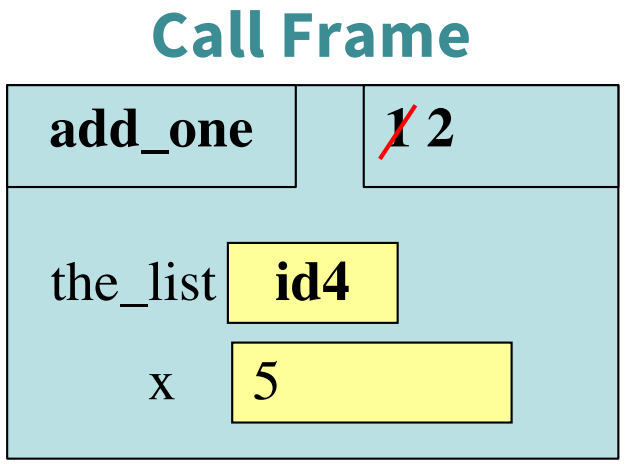


Modifying the Loop Variable (2)

```
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)
```





Modifying the Loop Variable (3)

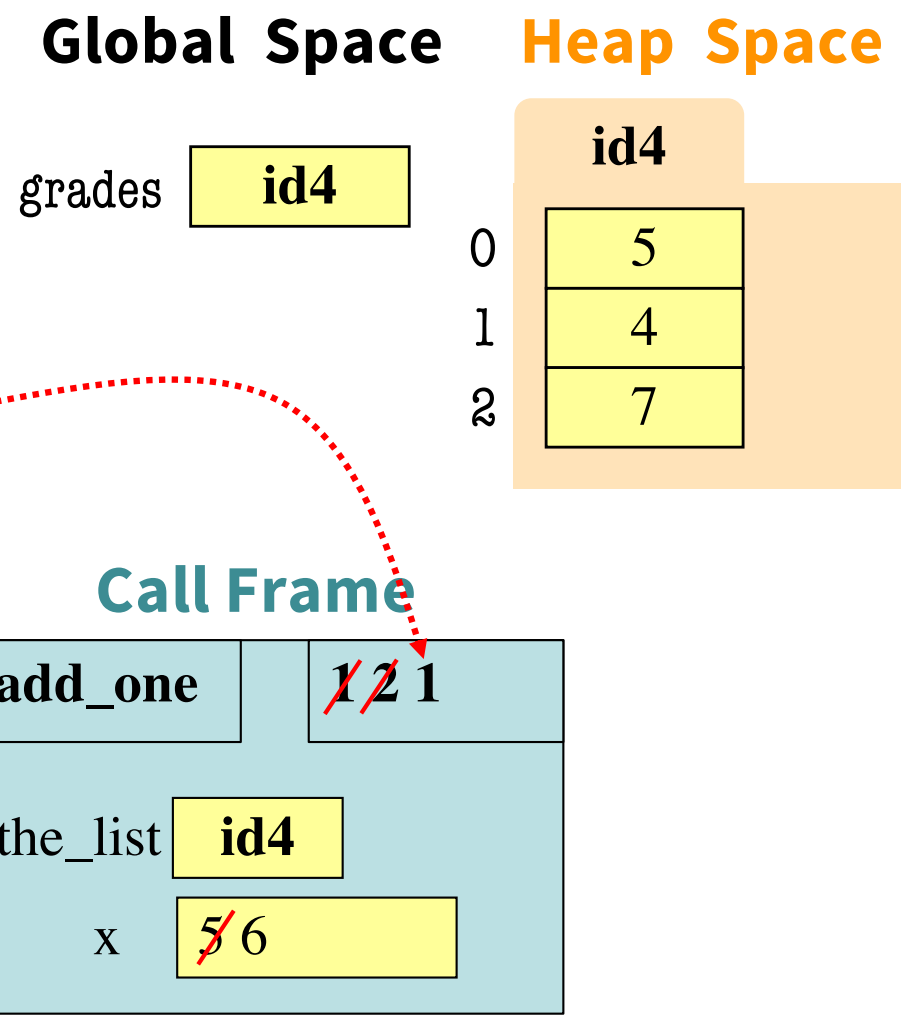
```
def add_one(the_list):
    """Adds 1 to every elt
    Pre: the_list is all numb."""
1   for x in the_list:
2       x = x+1
```

Loop back to line 1

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

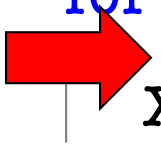
Increments x in **frame**

Does not affect folder



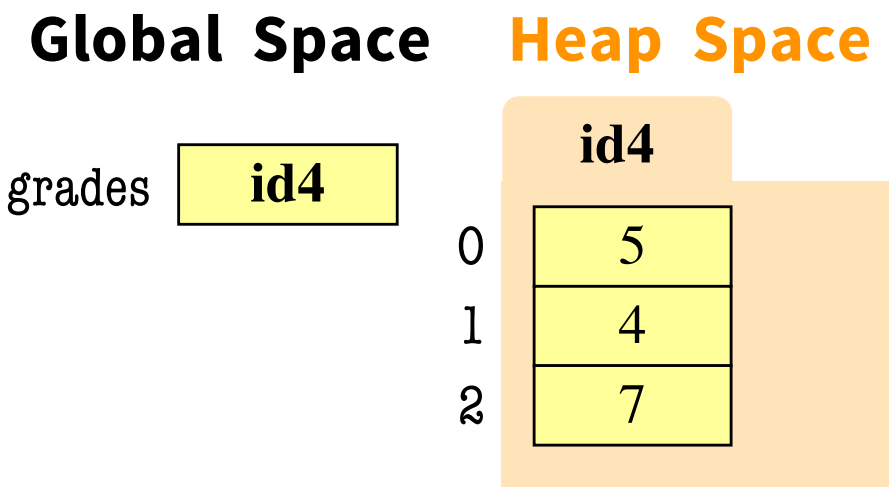
Modifying the Loop Variable (4)

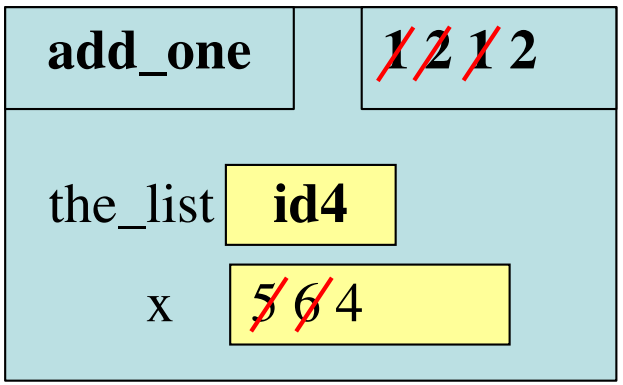
```
def add_one(the_list):
    """Adds 1 to every elt
    Pre: the_list is all numb."""
1   for x in the_list:
2       x = x+1
```



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

Next element stored in x.
Previous calculation lost.



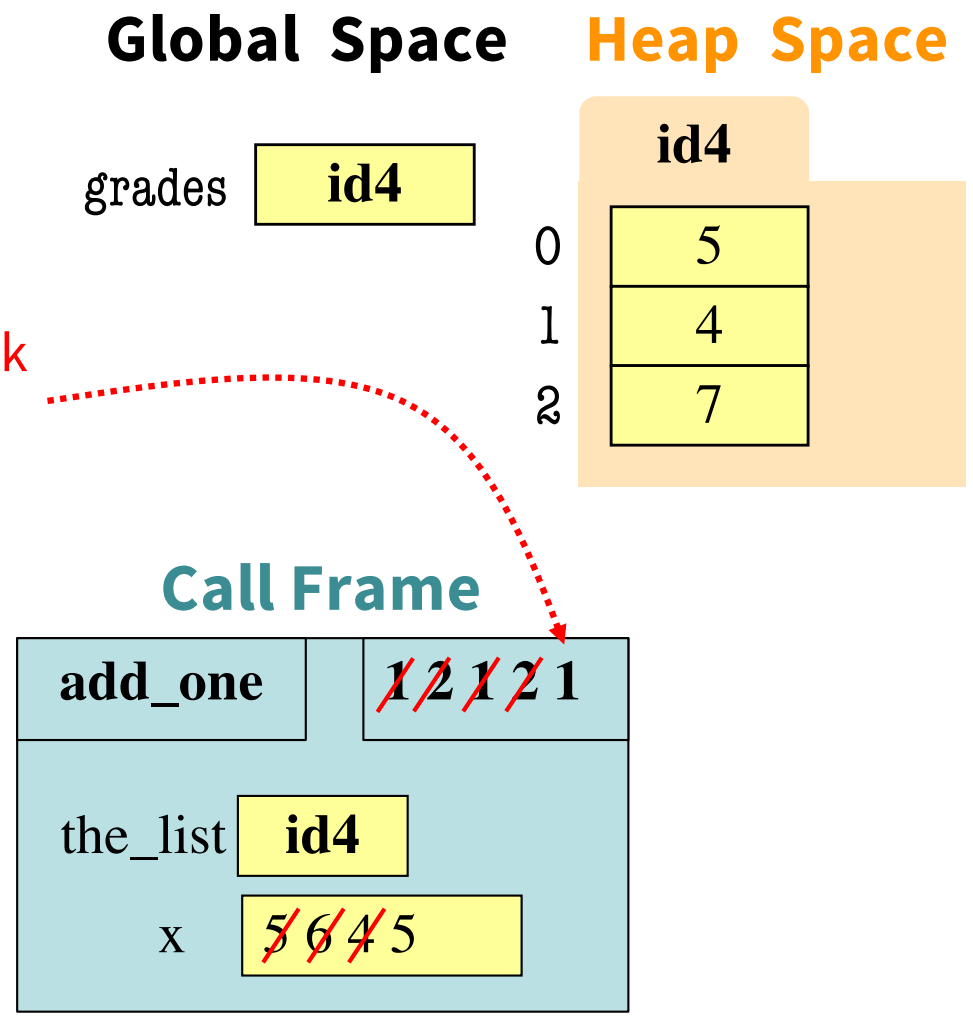


Modifying the Loop Variable (5)

```
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
```

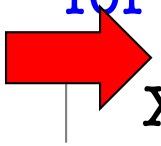
Loop back to line 1

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



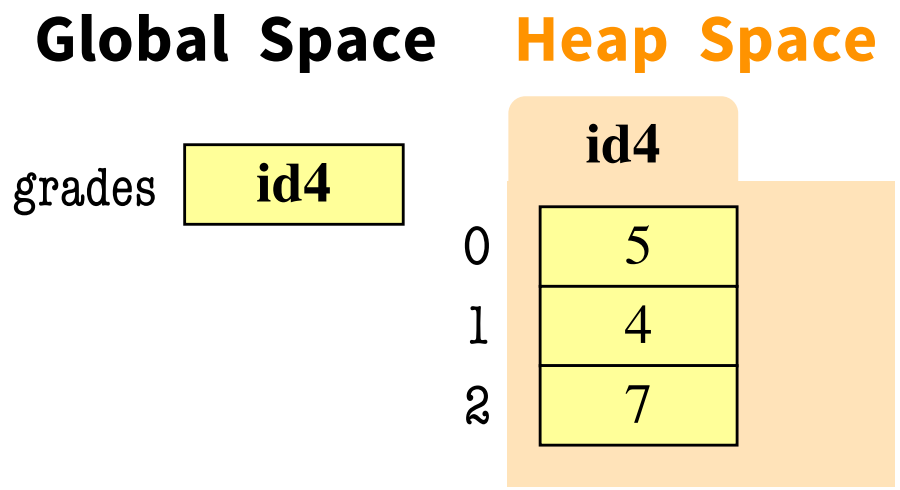
Modifying the Loop Variable (6)

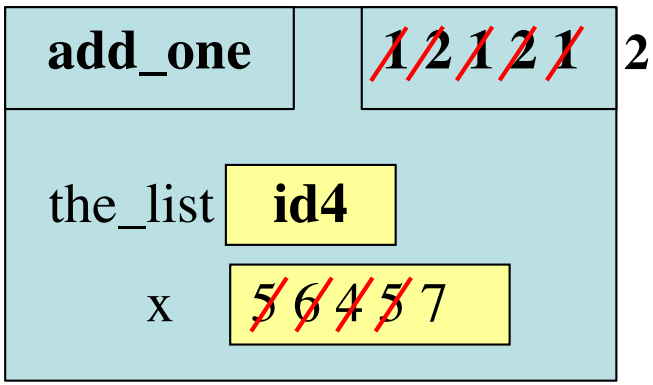
```
def add_one(the_list):
    """Adds 1 to every elt
    Pre: the_list is all numb."""
1   for x in the_list:
2       x = x+1
```



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

Next element stored in x.
Previous calculation lost.



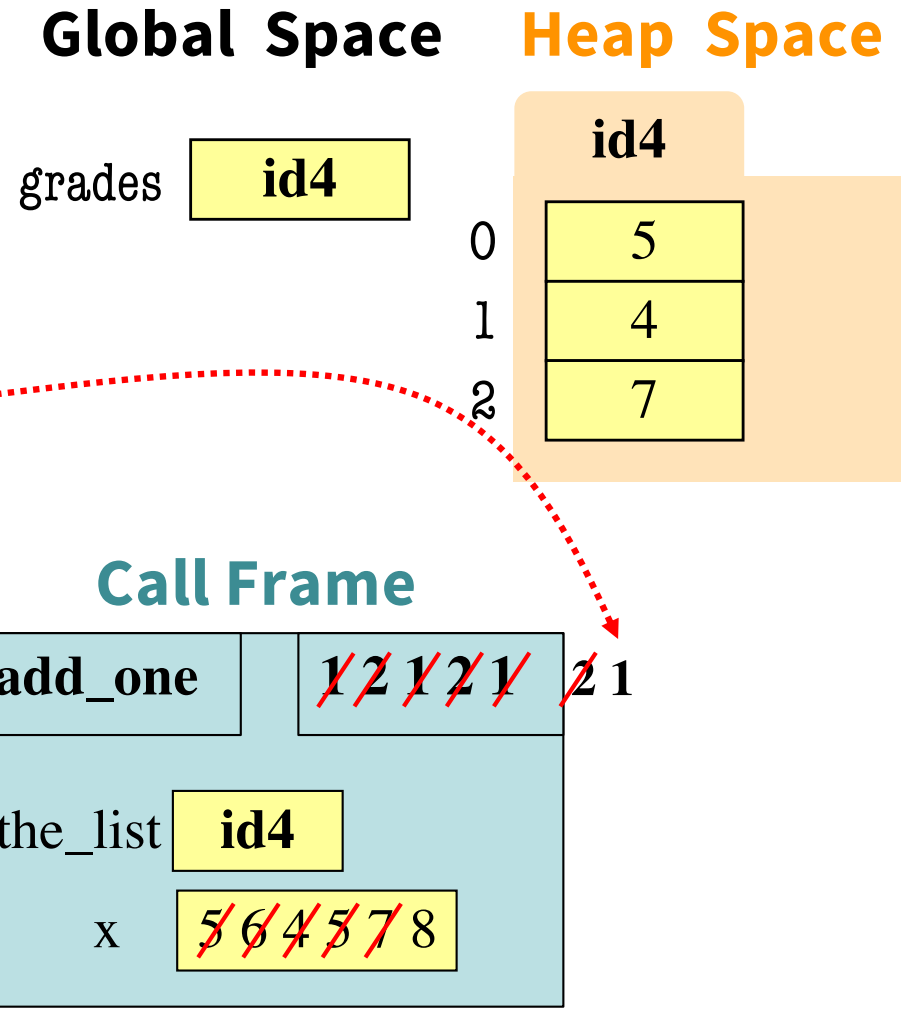


Modifying the Loop Variable (7)

```
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
```

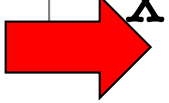
Loop back to line 1

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



Modifying the Loop Variable (8)

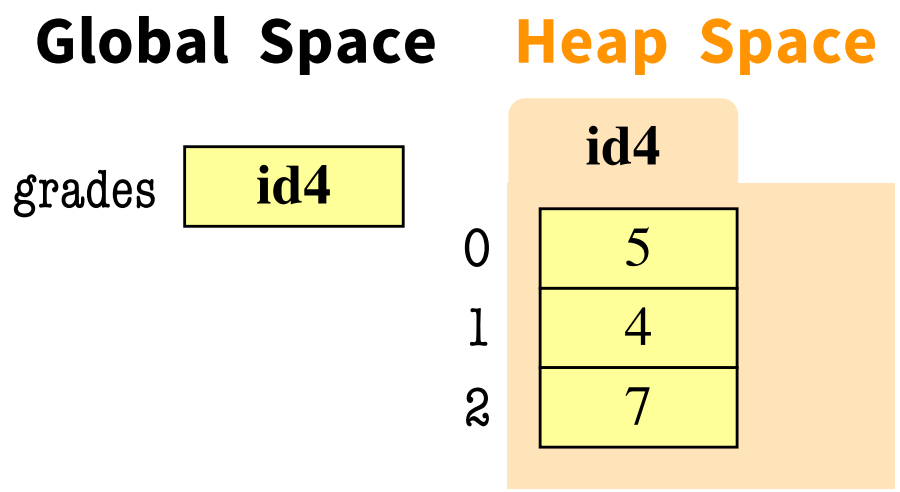
```
def add_one(the_list):
    """Adds 1 to every elt
    Pre: the_list is all numb."""
1   for x in the_list:
2       x = x+1
```

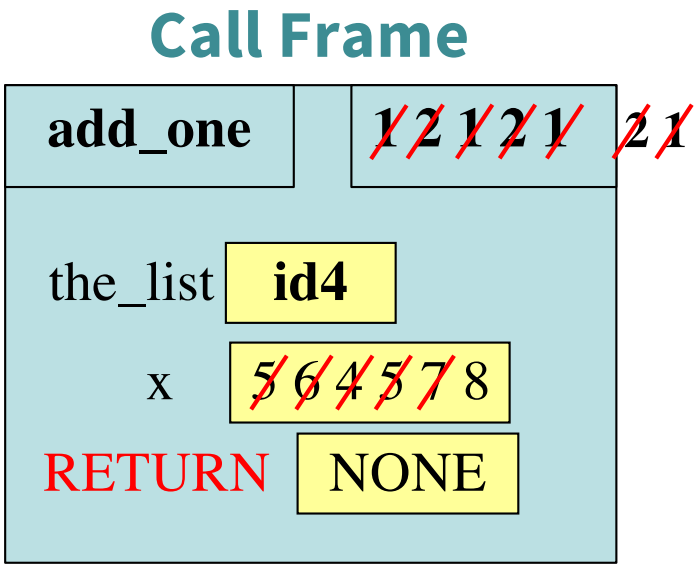


```
grades = [5,4,7]
add_one(grades)
```

Loop is **completed.**

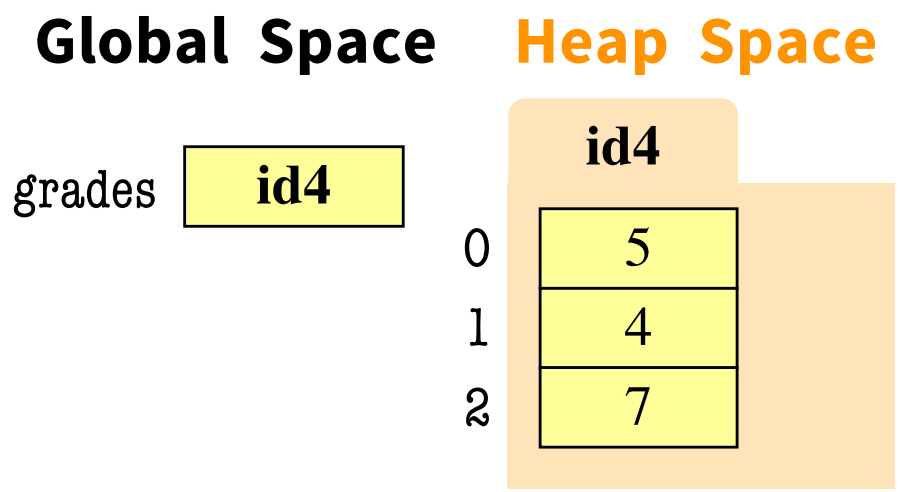
Nothing new put in x.



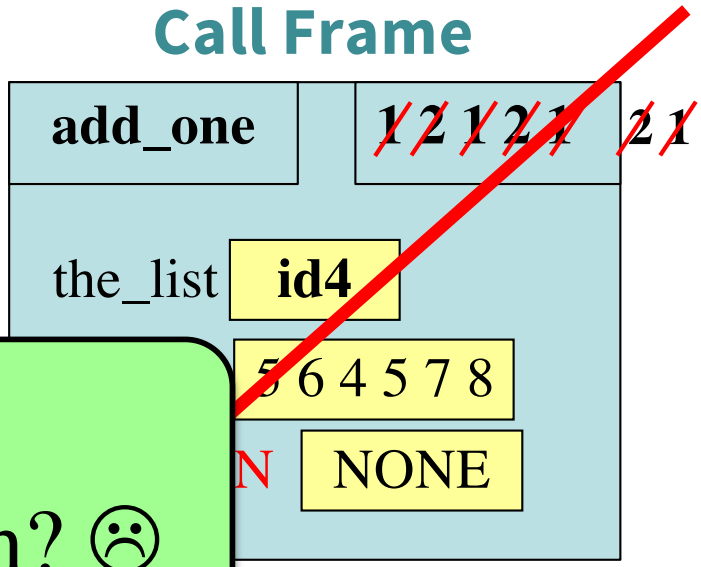


Modifying the Loop Variable (9)

```
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
```




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



No lasting changes.
What did we accomplish? ☹

Common For-Loop Mistakes (2)

Mistake #1: Modifying the loop variable instead of the list itself.

Mistake #2: Modifying the loop sequence as you walk through it.

For-Loop Mistake #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

For-Loop Mistake #2 (A)

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
```

INFINITE LOOP!

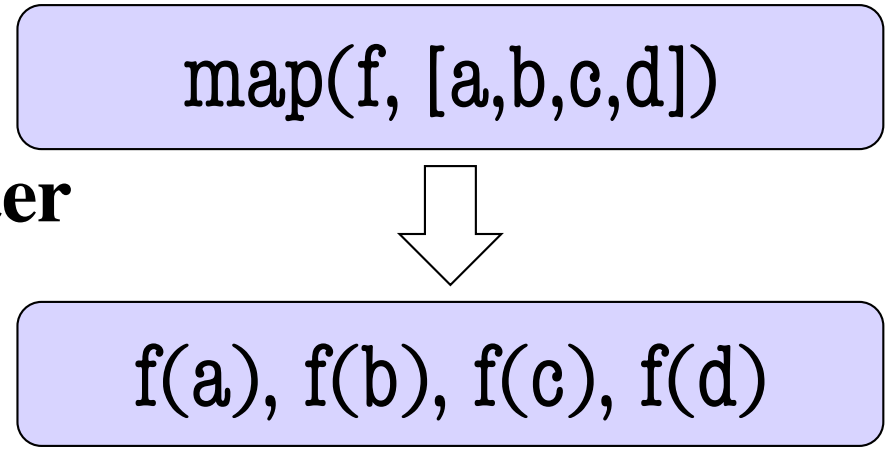
A: never prints b **CORRECT***
B: [1, 2, 3, 1, 2, 3]
C: [1, 2, 3]
D: I do not know

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

The Map Function

map(⟨*function*⟩, ⟨*list*⟩)

- ⟨*function*⟩ takes **1 parameter**
- Otherwise, error



Important: map does not return a list

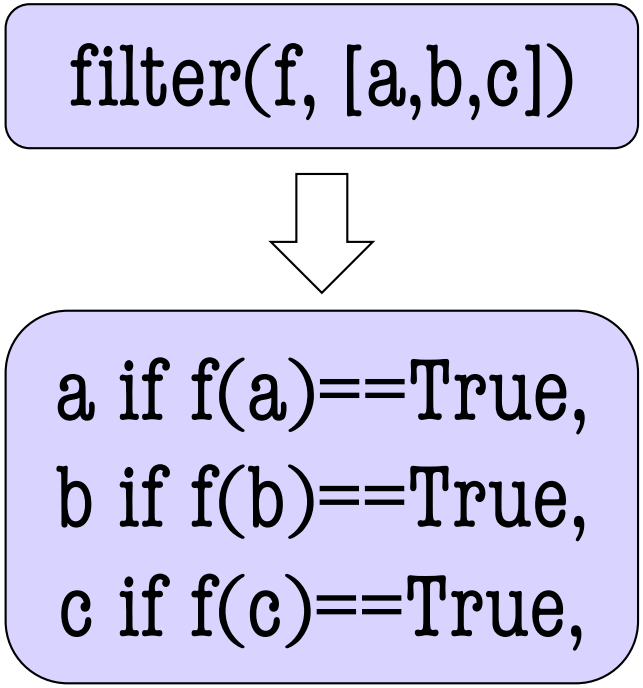
→ need to convert map's return value into a list

```
>>> len_list = list(map(len, ['a', 'bc', 'defg']))
>>> len_list
[1, 2, 4]
```

The Filter Function

filter(⟨*Boolean_function*⟩, ⟨*list*⟩)

- ⟨*function*⟩ takes **1 parameter**
- ⟨*function*⟩ returns a Boolean
- Collects elements of ⟨*list*⟩ for which ⟨*Boolean_function*⟩ returns True



Important: filter does not return a list

→ need to convert map's return value into a list

See ints.py to see filter in action