CS 111: Program Design I

Lecture 17: Lists, List memory model, List Methods, function variables, Web crawler law

Robert H. Sloan & Richard Warner
University of Illinois at Chicago
Thursday, October 24, 2019

UIC

LISTS

Next big project: Finish Web Crawler

- We will need to make more extensive use of lists than we have up until now
- Recall our strategy something like

Crawl all pages reachable from start

- List of pages to visit, initially start
- while that list is not empty:
  - Take a page from the list
  - Get its text # need to learn how to do this
  - remove that page from to-visit list, add it to already-visited list
  - Get all the links in that page
  - for each link
    - if not in already-visited list
    - add it to to-visit list

And besides

- Lists in Python are just awesome

Recall

- Lists are data structures that let us store collections of data in sequence with indices

```
A = [2, 3, 5]
B = ['Brennan']
C = A + B
```

print(C) will result in:

A. [2, 3, 5]

B. [2, 3, 5, 'Brennan']

C. ['2', '3', '5', 'Brennan']

D. This will cause an error

E. I don't know

Remember the smallest

- '' # Empty string, could also be written ""
- [] # empty list

List *Functions*

- *Not* methods*; there are also list methods,* e.g., append
- len: length of list (i.e., number of elements)
- + will concatenate lists
- min, max: minimum or maximum of list
- sum: sum of the elements in the list
  - E.g., sum([2, 3, 5]) → 10

Important: Lists are **mutable**

```
>>> years= [1788, 1800, 1860, 1932]
>>> years[0] = years[0] + 4
>>> print(years)
[1792, 1800, 1860, 1932]
```

Lists versus Strings

List	String
Elements can be any type	Elements are characters
Mutable	Immutable
Heterogeneous elements	Homogenous elements
Can be nested in other lists	

What is printed

```
lst = ['abc.com', 'cnn.com', 'msnbc.com']
lst[1] = 'fox.com'
print(len(lst))
```

A. 3
B. 4
C. 23
D. 30
E. No output; error in 2nd line of code

What is printed

```
lst = ['abc.com', 'cnn.com', 'msnbc.com']
lst[1] = 'fox.com'
now lst has become ['abc.com', 'fox.com', 'msnbc.com']
print(len(lst))
```

A. 3
B. 4
C. 23
D. 30
E. No output; error in 2nd line of code

Reminder: Compound (assignment) operators

- Reminder that Python has compound operators +=, -=, *=, /=, and %=
- Does the operation and then assigns
- age += 1 # is shorthand for
- age = age + 1
- fun *= 2 # is shorthand for
- fun = fun * 2
- N.B.: Variable on left can even refer to immutable object!

SCOPE; MEMORY MODEL FOR LISTS

What will this output?

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
            ls[i] += 1

ls = [2, 6, 7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```

Clicker	print(ls, " ", x)
A	[2,6,7] 5
B	[3,7,8] 6
C	[3,7,8] 5
D	[2,6,7] 6
E	I don't know

What will this output?

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
            ls[i] += 1

ls = [2, 6, 7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```

Clicker	print(ls, " ", x)
A	[2,6,7] 5
B	[3,7,8] 6
C	[3,7,8] 5
D	[2,6,7] 6
E	I don't know

Parameter passing

- Assignment statement *inside* function creates local variable
  - Distinct object from any outside function; exists only inside function; can't be used outside
- And formal parameters of functions are also local variables
- That's all there is to think about and all that's worth knowing about local variables
  - *As long as only reference to immutable object passed in as actual parameter*

Still, a little slower: Local vs. Global variables

- Scope: Region of program where identifier (e.g., variable name) valid and can be used to refer to object. Environment where variable name is evaluated
  - More precisely/advanced: region of program where name–entity binding is valid
- Local variable defined inside function; scope only inside that function
- Global variable created (by assignment statement) outside any function available everywhere

Use Global Variables Sparingly

- When in doubt; use local variables. Avoids confusion over meaning of name
- Example of when global *is* appropriate: constant (variable whose value should never change)

```
ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
CARDS_IN_DECK = 52
c = 186282 # speed of light in mph
```

Inside function

- Function looks up variables in order
  1. Local variables (including parameters)
  2. Global variables (often bad style)

Concept check

```
def f(num):
    num = 4
    return 2

def g(val):
    num = 8
    print(f(1))

g(2)
```

What gets printed?

A. 1
B. 2
C. 4
D. 8
E. Error because undefined variable

Concept check

```
def f(num):
    num = 4
    return 2

def g(val):
    val = 8
    print(f(1))

g(2)
```

What gets printed?

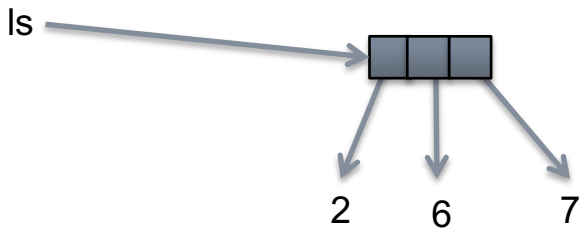
A. 1
B. 2
C. 4
D. 8
E. Error because undefined variable

Parameter passing: *lists* as arguments (actual parameters)

- Passing list as argument to function passes reference to that list, *not a copy*
  - Changes made by function will be visible afterwards in caller's scope!
  - **Recall: Scope**: environment in which variable evaluated
- Functions that change lists passed to them as parameters called modifiers; changes they make called side effects (of calling function)

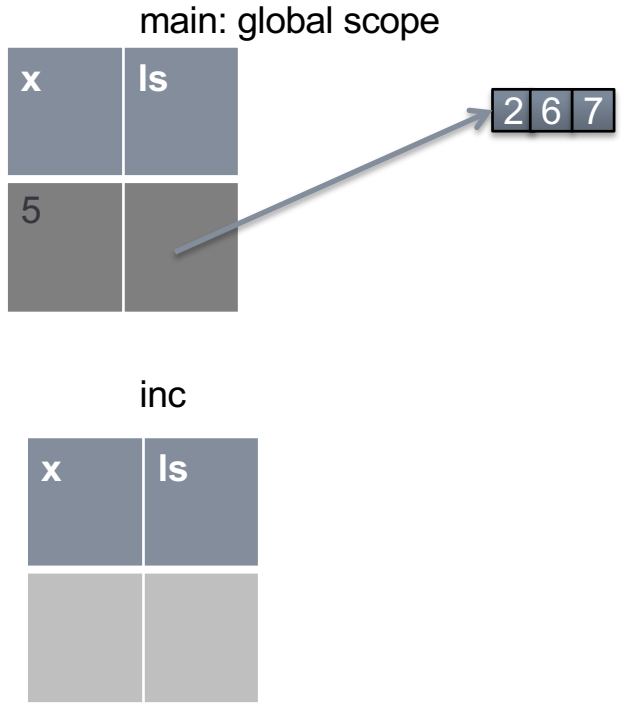
Memory diagram of a list

- ls = [2, 6, 7]
- Technically every list's name is reference to collection of references and we should really draw our memory diagram like upper figure
- When all elements of immutable types (a common case) will usually use simple lower picture



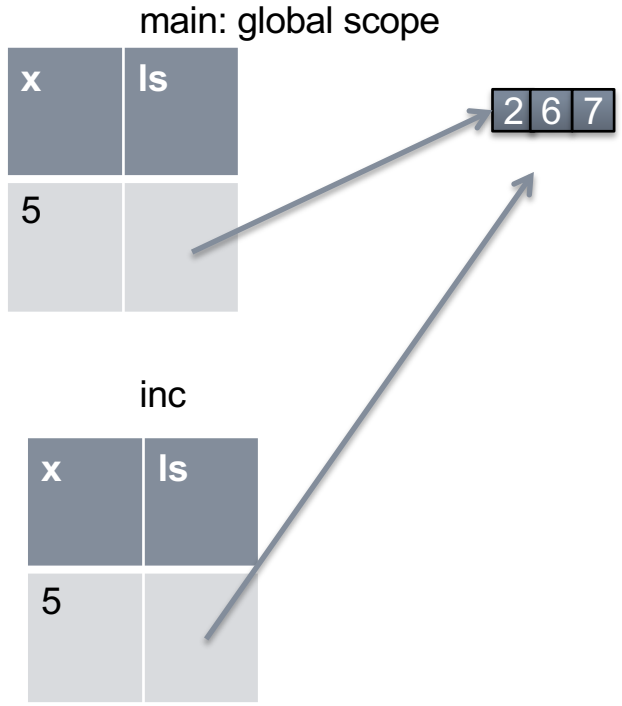
Memory diagram of clicker question

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
        ls[i] += 1
ls = [2,6,7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```



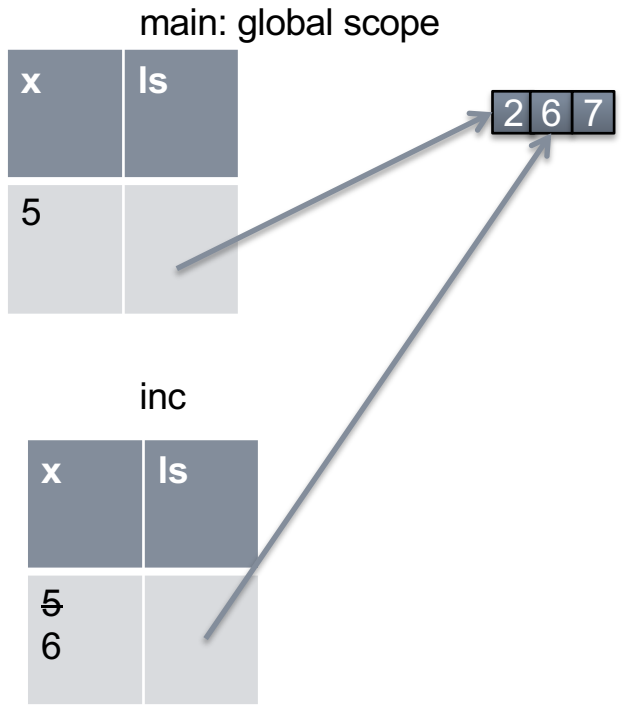
Memory diagram of clicker question

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
        ls[i] += 1
ls = [2,6,7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```



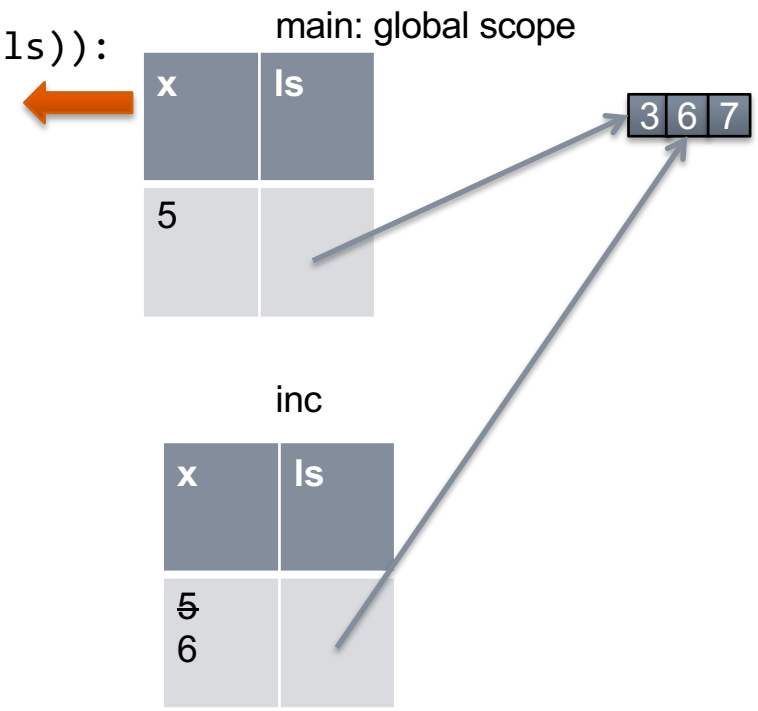
Memory diagram of clicker question

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
        ls[i] += 1
ls = [2,6,7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```



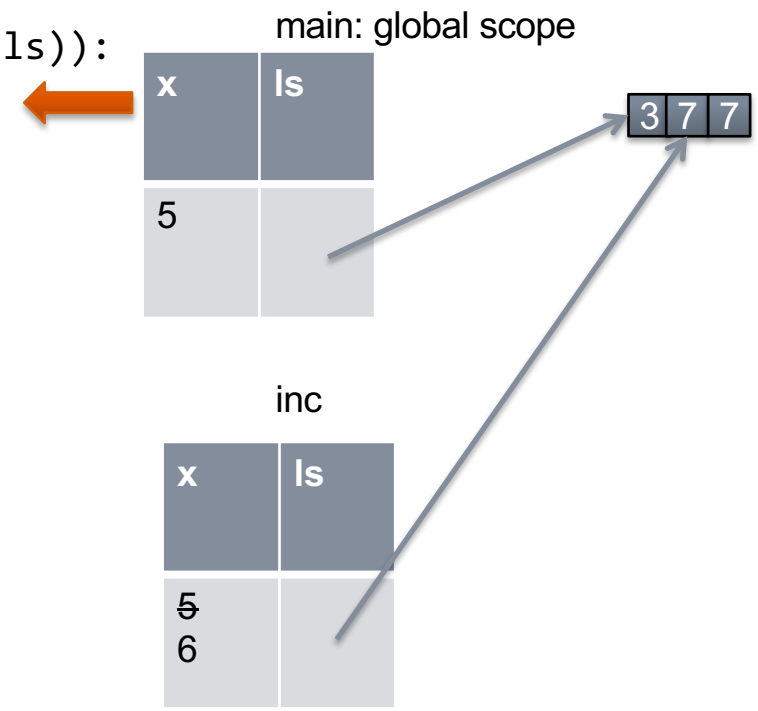
Memory diagram of clicker question

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
        ls[i] += 1
ls = [2,6,7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```



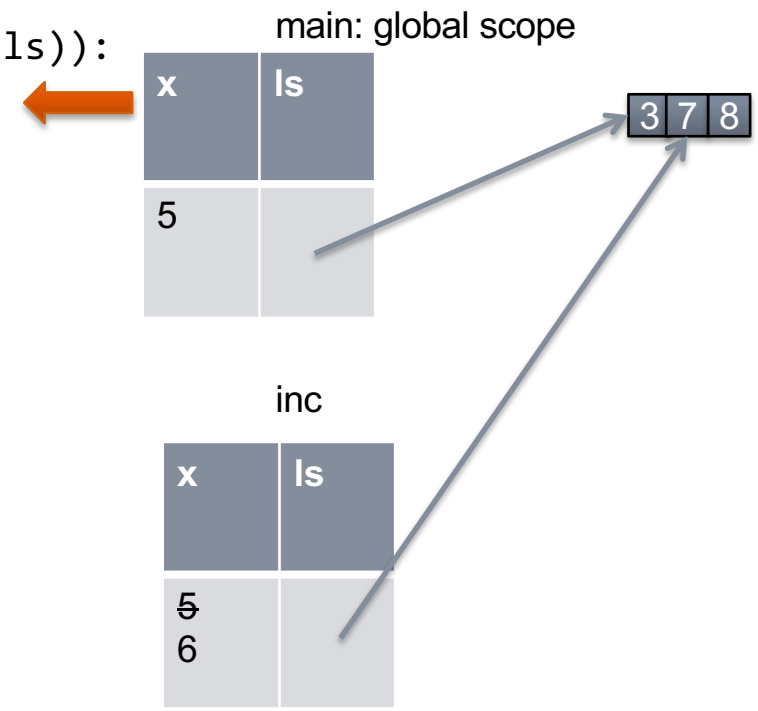
Memory diagram of clicker question

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
        ls[i] += 1
ls = [2,6,7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```



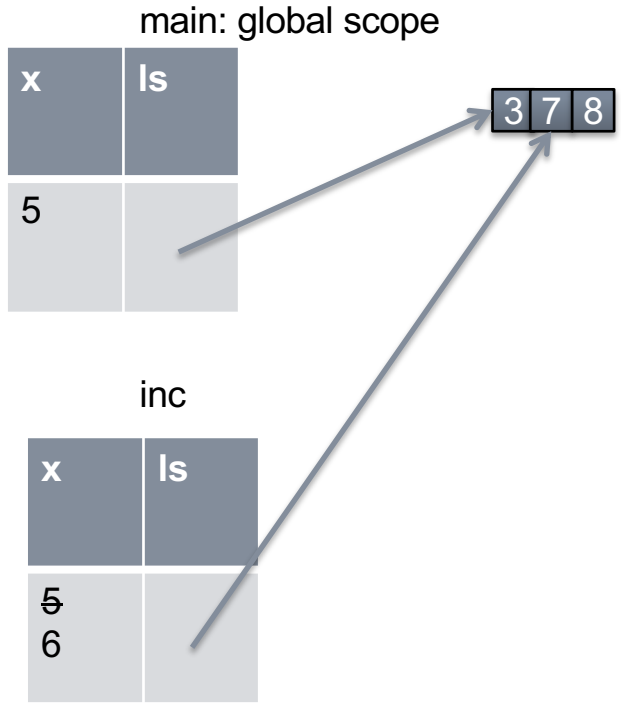
Memory diagram of clicker question

```
def inc(ls, x):
    x += 1
    for i in range(len(ls)):
        ls[i] += 1
ls = [2,6,7]
x = 5
inc(ls, x)
print(ls, ' ', x)
```



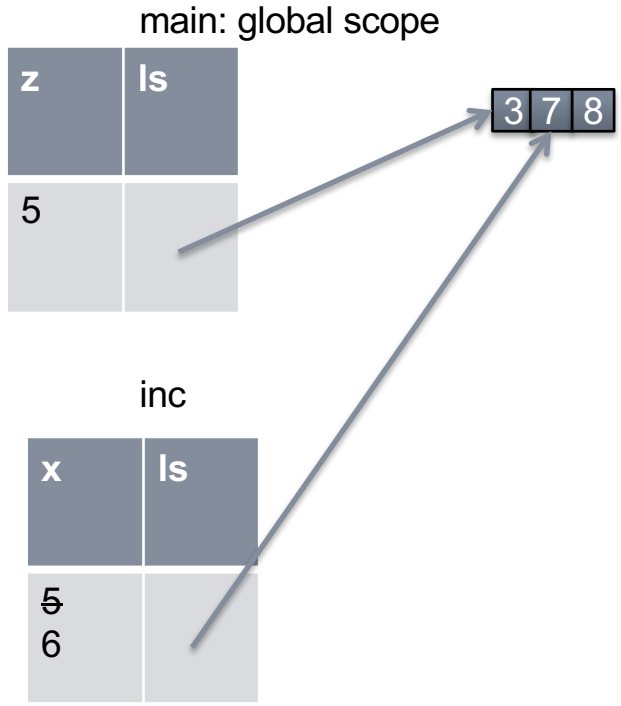
Memory diagram of clicker question

```
def inc(ls, x):
    x = x + 1
    for i in range(len(ls)):
        ls[i] = ls[i] + 1
ls = [2,6,7]
x = 5
inc(ls,x)
print(ls, ' ', x)
```



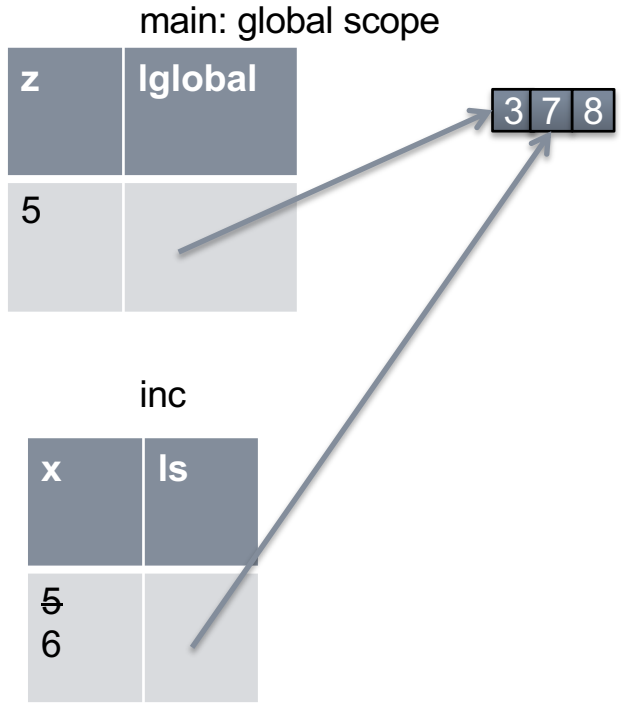
Memory diagram of clicker question

```
def inc(ls, x):
    x = x + 1
    for i in range(len(ls)):
        ls[i] = ls[i] + 1
ls = [2,6,7]
z = 5
inc(ls,z)
print(ls, ' ', z)
```



Memory diagram of clicker question

```
def inc(ls, x):
    x = x + 1
    for i in range(len(ls)):
        ls[i] = ls[i] + 1
lglobal = [2,6,7]
z = 5
inc(lglobal, z)
print(lglobal, ' ', z)
```



LIST METHODS

Source of list methods material

- Much of this material based on but modified from "CS1 in Python Peer Instruction Materials" by Daniel Zingaro, in repository http://www.peerinstruction4cs.org/ licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. (License at: https://creativecommons.org/licenses/by-nc-sa/3.0/deed.en_US)

Methods: Getting info

- You can learn about methods for standard built-in types, e.g., string, list, in
  - Python documentation https://docs.python.org/3/library/index.html
    - For many list functions *and* methods, see Sequence Types → Mutable sequence types
- Recall at Python console dir(list), dir(str), etc. tells names of all methods
  - For CS 111, ignore all methods with names starting with underscore; we won't use

Key list methods

- ls.append(item): add item to end of ls
- ls.pop(): remove and return element at end of ls
- ls.pop(i): remove and return element at index i in ls
- ls.remove(item): remove first occurrence of item from ls
- ls.insert(i, item): insert item into ls at position i
  - sliding elements of ls[i:] all one position right to make room
- ls.sort(): Move elements of ls so that ls is in sorted order
  - Requires all elements to be comparable

- *All those methods modify* ls
- *Only* pop *among those methods has a return value*