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

Lecture 10: Lists and Sequences

(Sections 10.0-10.2, 10.4-10.6, 10.8-10.13)

CS 1110

Introduction to Computing Using Python

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

Sequences: Lists of Values

String

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Put characters in quotes
  - Use \' for quote character
- Access characters with []
  - s[0] is 'a'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'

List

- x = [5, 6, 5, 9, 15, 23]

0	1	2	3	4	5
5	6	5	9	15	23

- Put values inside [ ]
  - Separate by commas
- Access **values** with []
  - x[0] is 5
  - x[6] causes an error
  - x[0:2] is [5, 6] (excludes 2nd 5)
  - x[3:] is [9, 15, 23]

Sequence is a name we give to both

Lists Have Methods Similar to String

```
x = [5, 6, 5, 9, 15, 23]
```

- `<list>.index(<value>)`
  - Return position of the value
  - **ERROR** if value is not there
  - `x.index(9)` evaluates to 3
- `<list>.count(<value>)`
  - Returns number of times value appears in list
  - `x.count(5)` evaluates to 2

But to get the length of a list you use a function, not a class method:

`len(x)`

~~`x.len()`~~

Things that Work for All Sequences

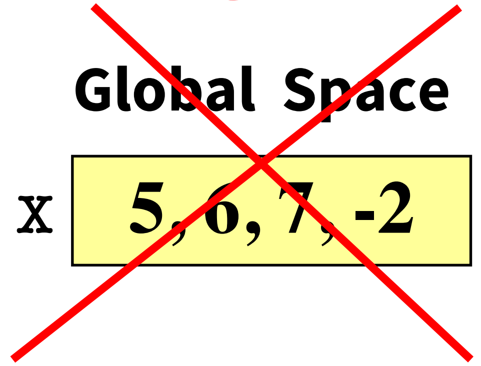
s = 'slithy'

x = [5, 6, 9, 6, 15, 5]

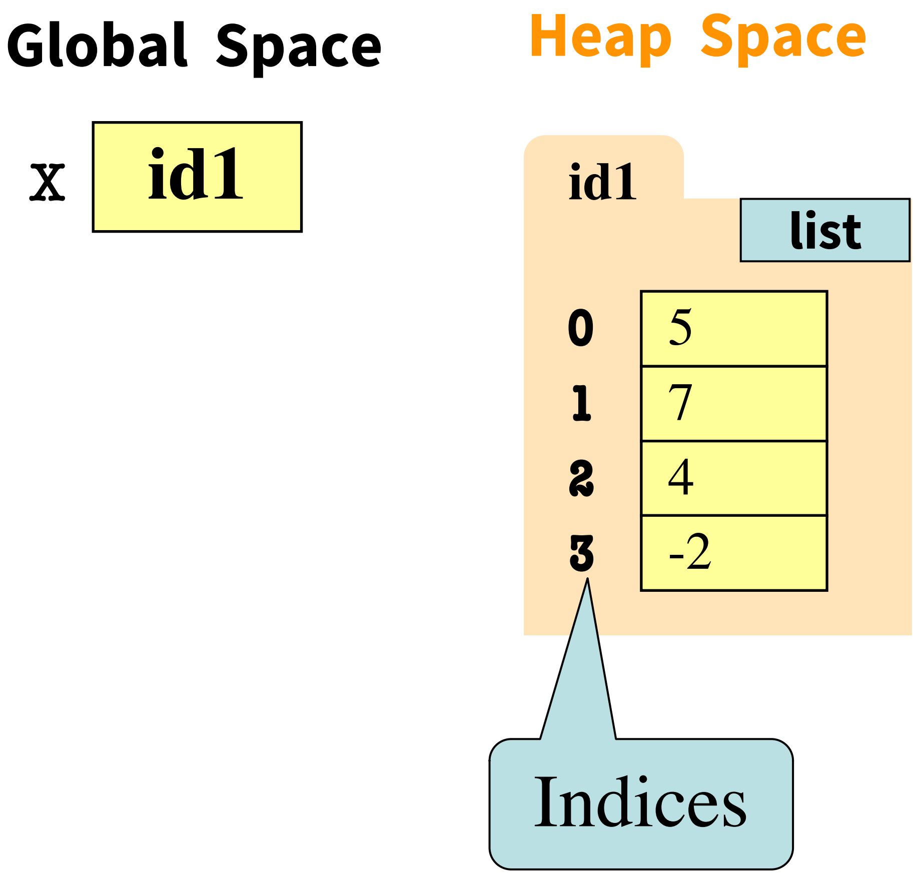
s = 'slithy'		x = [5, 6, 9, 6, 15, 5]
s.index('s') → 0	methods	x.index(5) → 0
s.count('t') → 1	methods	x.count(6) → 2
len(s) → 6	built-in fns	len(x) → 6
s[4] → "h"	slicing	x[4] → 15
s[1:3] → "li"	slicing	x[1:3] → [6, 9]
s[3:] → "thy"	slicing	x[3:] → [6, 15, 5]
s[-2] → "h"	slicing	x[-2] → 15
s + ' toves' → "slithy toves"	operators	x + [1, 2] → [5, 6, 9, 6, 15, 5, 1, 2]
s * 2 → "slithyslithy"	operators	x * 2 → [5, 6, 9, 6, 15, 5, 5, 6, 9, 6, 15, 5]
't' in s → True	operators	15 in x → True

Representing Lists

Wrong:



Correct:



```
x = [5, 7, 4,-2]
```

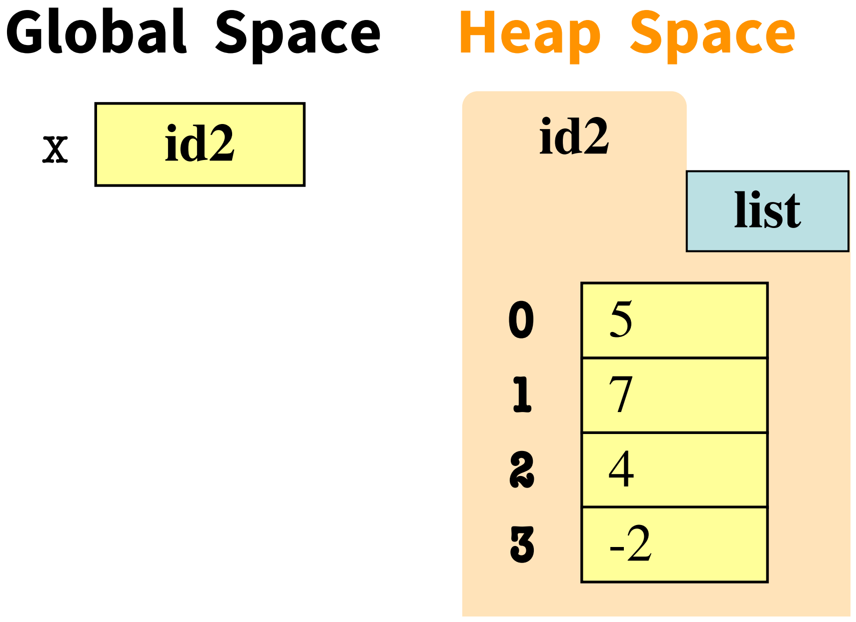
Lists vs. Class Objects

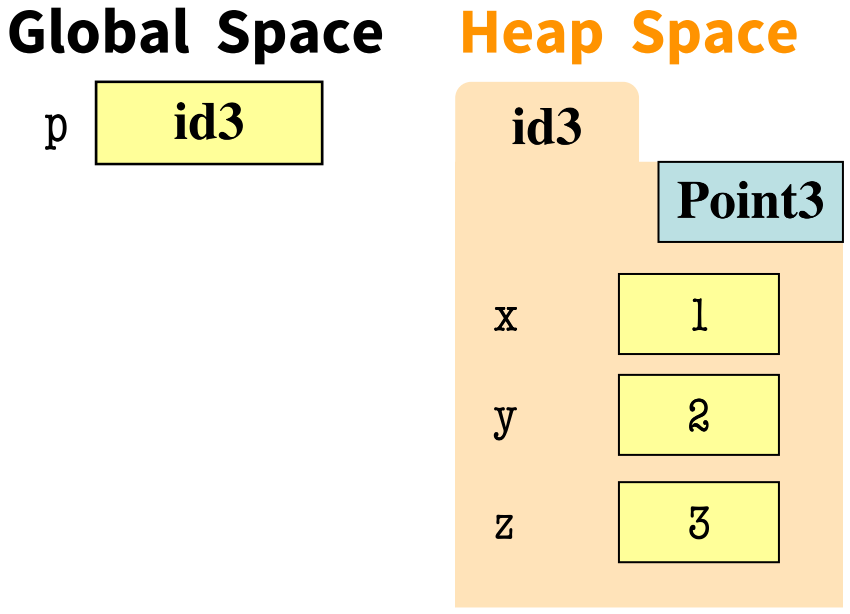
List

- Attributes are indexed
  - Example: x[2]

Objects

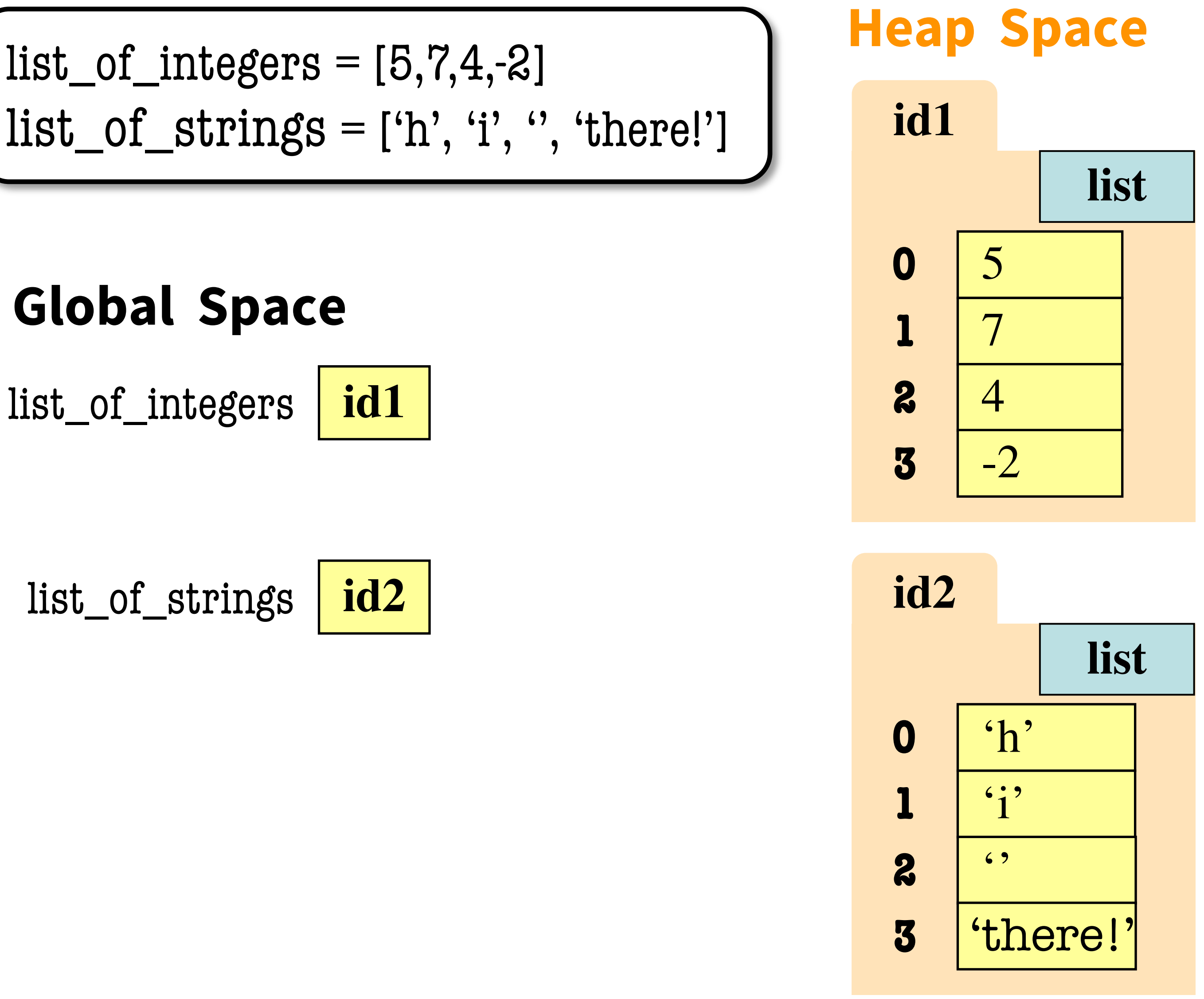
- Attributes are named
  - Example: p.x





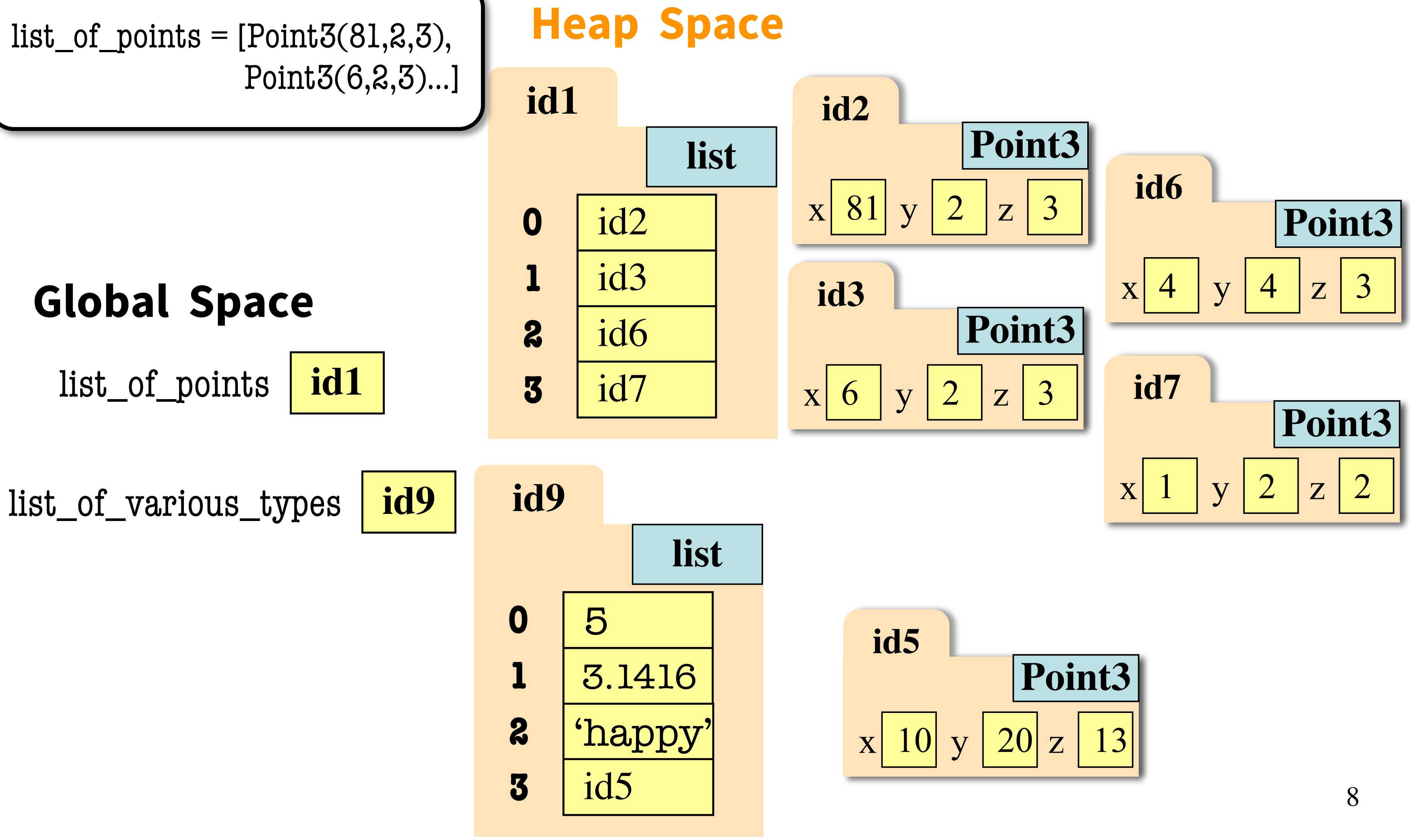
Lists Can Hold Any Type

```
list_of_integers = [5,7,4,-2]
list_of_strings = ['h', 'i', '', 'there!']
```



No Really, Lists Can Hold Any Type!

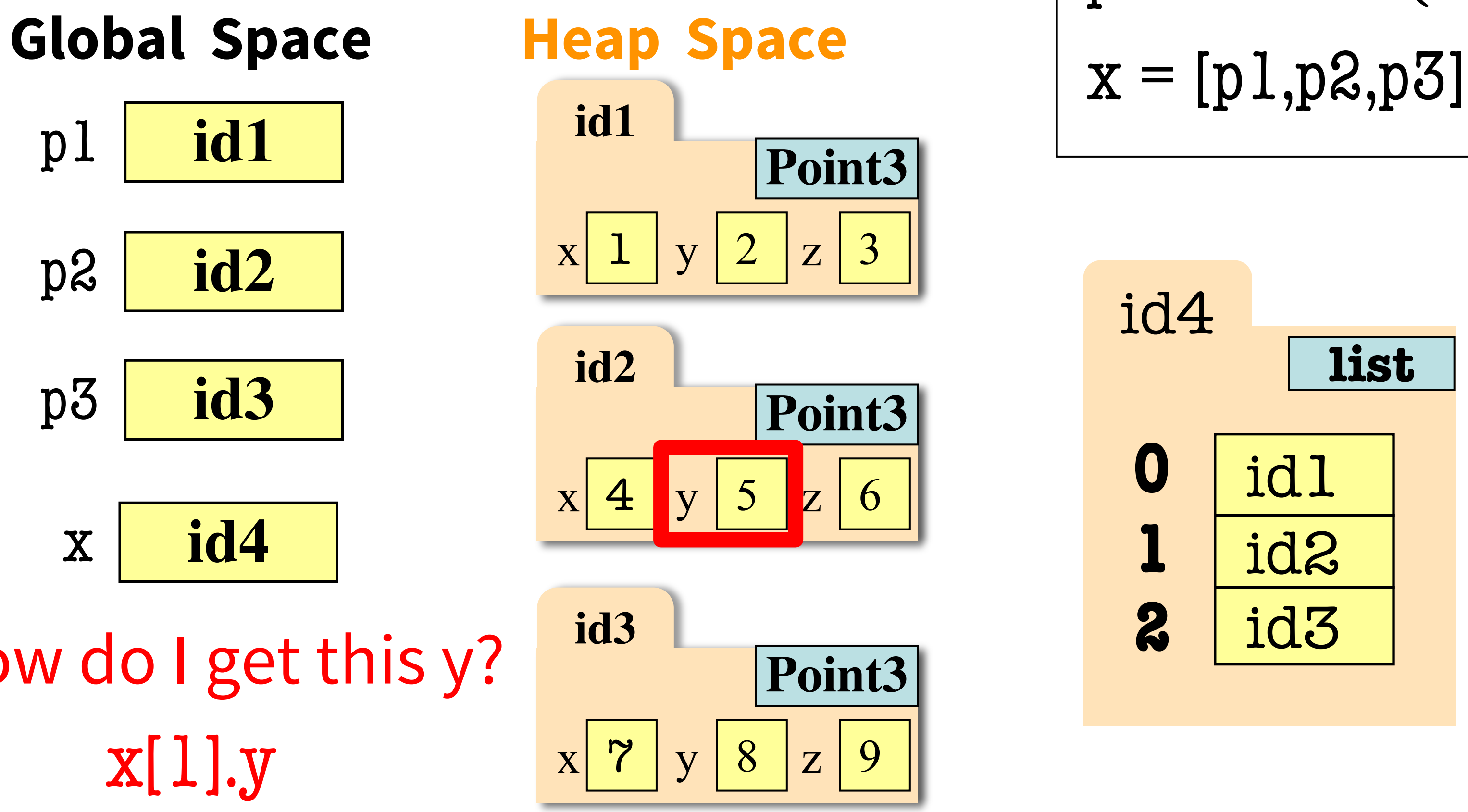
```
list_of_points = [Point3(81,2,3),
                  Point3(6,2,3)...]
```



Lists of Objects

- List elements are variables
  - Can store base types and ids
  - Cannot store folders

```
p1 = Point3(1, 2, 3)
p2 = Point3(4, 5, 6)
p3 = Point3(7, 8, 9)
x = [p1,p2,p3]
```



How do I get this y?

x[1].y

List Assignment

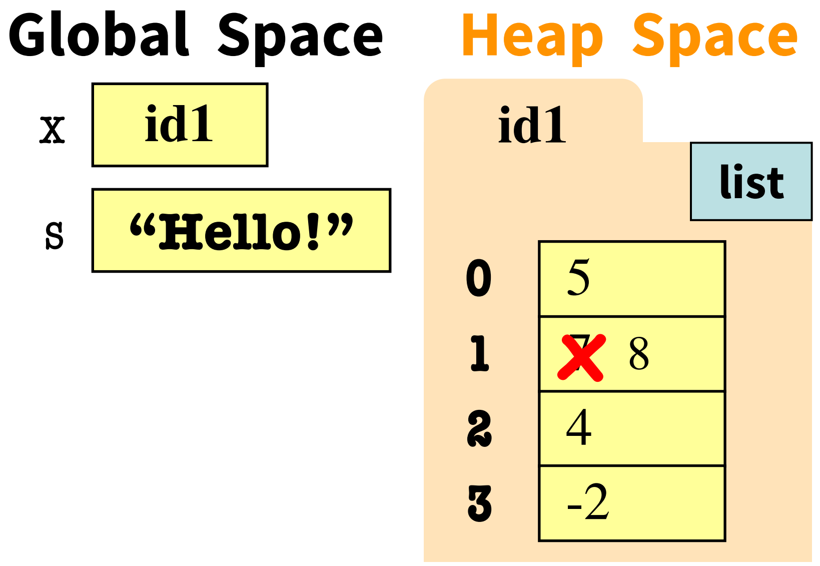
- **Format**:

  <var>[<index>] = <value>

  - Reassign at index
  - Affects folder contents
  - Variable is unchanged

- Strings cannot do this
  - Strings are **immutable**

```
x = [5, 7,4,-2]
x[1] = 8
s = "Hello!"
s[0] = 'J'
TypeError: 'str' object does
not support item assignment
```



List Methods Can Alter the List

```
x = [5, 6, 5, 9]
```

See Python API for more

- `<list>.append(<value>)`
  - Adds a new value to the end of list
  - `x.append(-1)` *changes* the list to `[5, 6, 5, 9, -1]`
- `<list>.insert(<index>,<value>)`
  - Puts value into list at index; shifts rest of list right
  - `x.insert(2,-1)` *changes* the list to `[5, 6, -1, 5, 9]`
- `<list>.sort()` What do you think this does?

1st Clicker Question

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> x[3] = -1
>>> x.insert(1, 2)
```

- What is `x[4]`?

A: 10
B: 9
C: -1
D: **ERROR**
E: I don't know

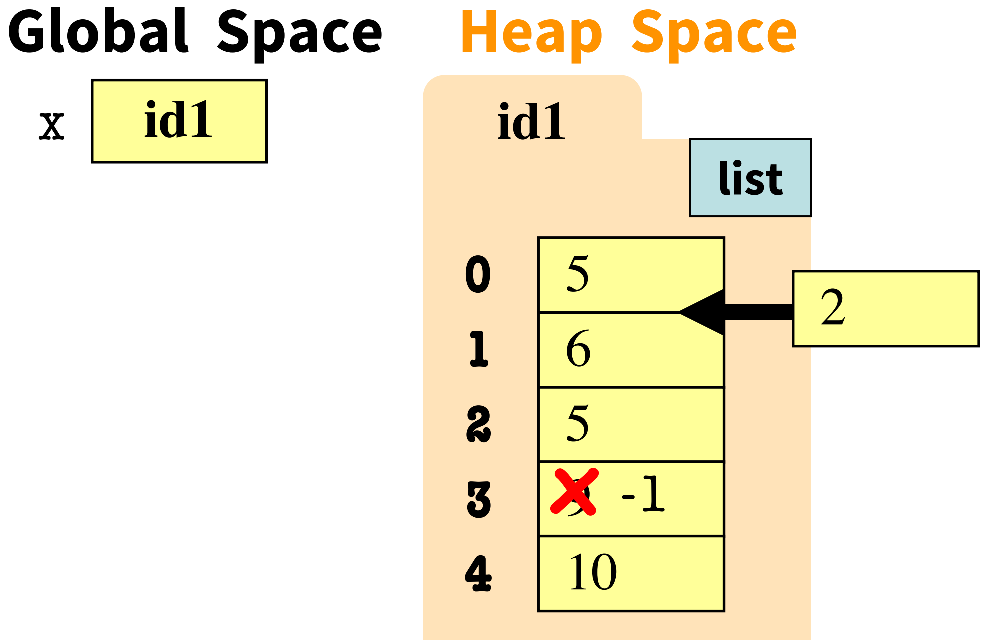
1st Clicker Answer

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> x[3] = -1
>>> x.insert(1, 2)
```

- What is x[4]?

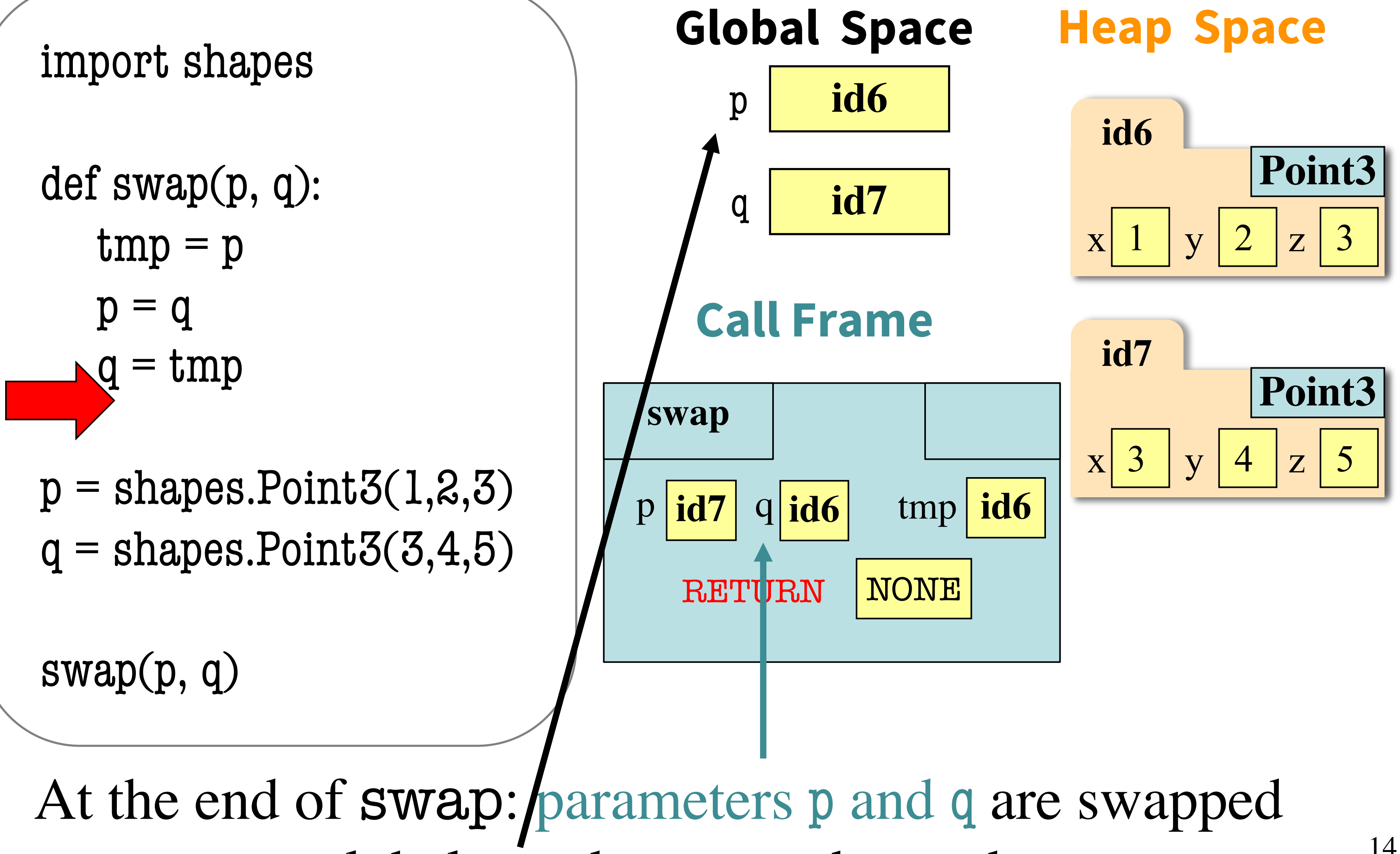
A: 10
B: 9
C: -1 **CORRECT**
D: **ERROR**
E: I don't know



(Original elements 1-4 are shifted down to be elements 2-5)

Recall: identifier assignment → no swap

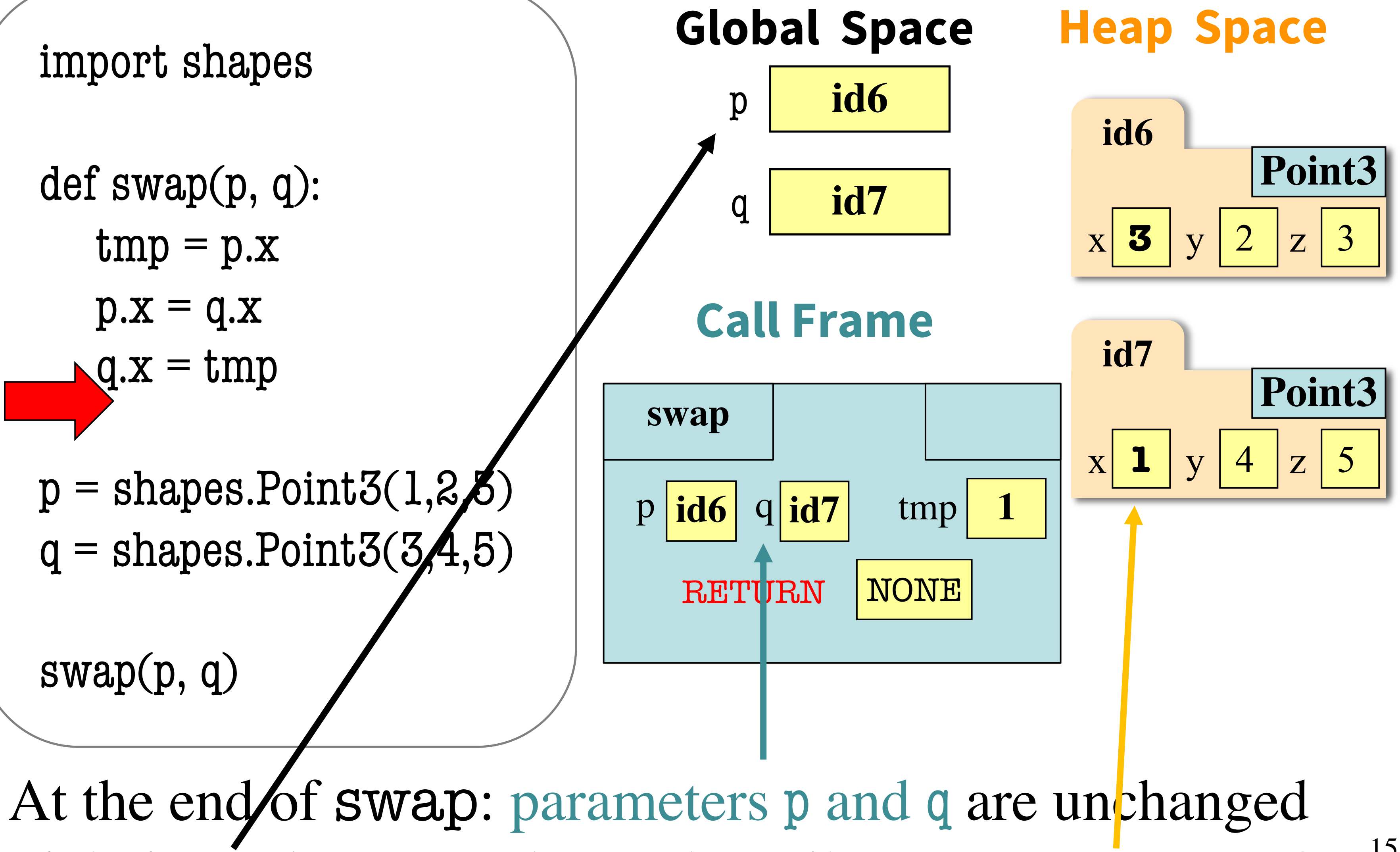
```
import shapes

def swap(p, q):
    tmp = p
    p = q
    q = tmp

p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

swap(p, q)
```



At the end of swap: parameters p and q are swapped
global p and q are unchanged

Recall: Attribute Assignment → swap!

```
import shapes

def swap(p, q):
    tmp = p.x
    p.x = q.x
    q.x = tmp

p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

swap(p, q)
```

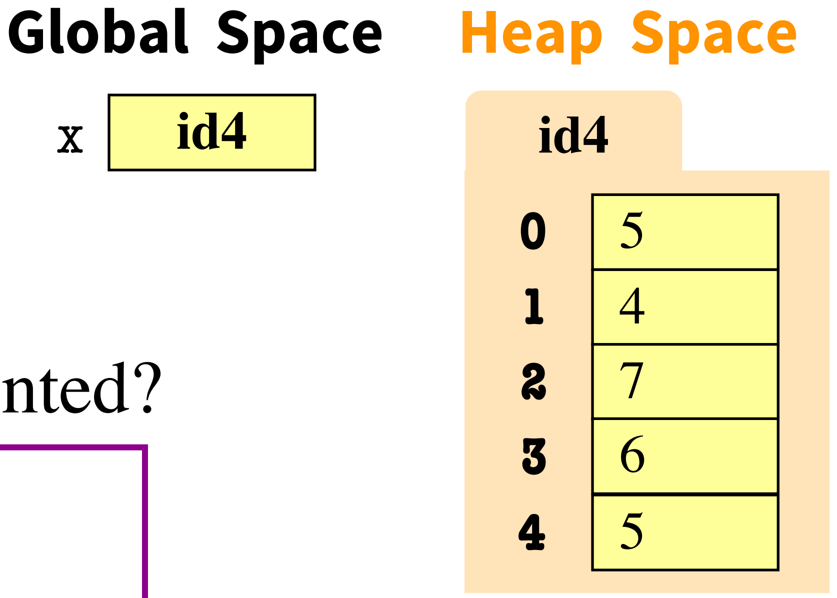


At the end of swap: parameters p and q are unchanged
global p and q are unchanged, attributes x are swapped

2nd Clicker Question

```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list"""
    temp= b[h]
    b[h]= b[k]
    b[k]= temp
```

```
x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```



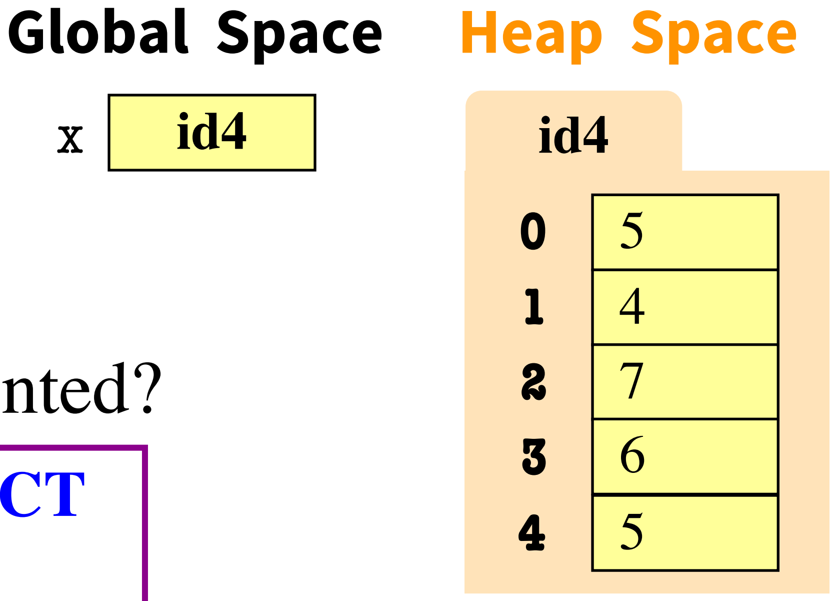
What gets printed?

A: 5
B: 6
C: Something else
D: I don't know

2nd Clicker Answer

```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list"""
    temp= b[h]
    b[h]= b[k]
    b[k]= temp

x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```

Swaps b[h] and b[k], because parameter b contains name of list.



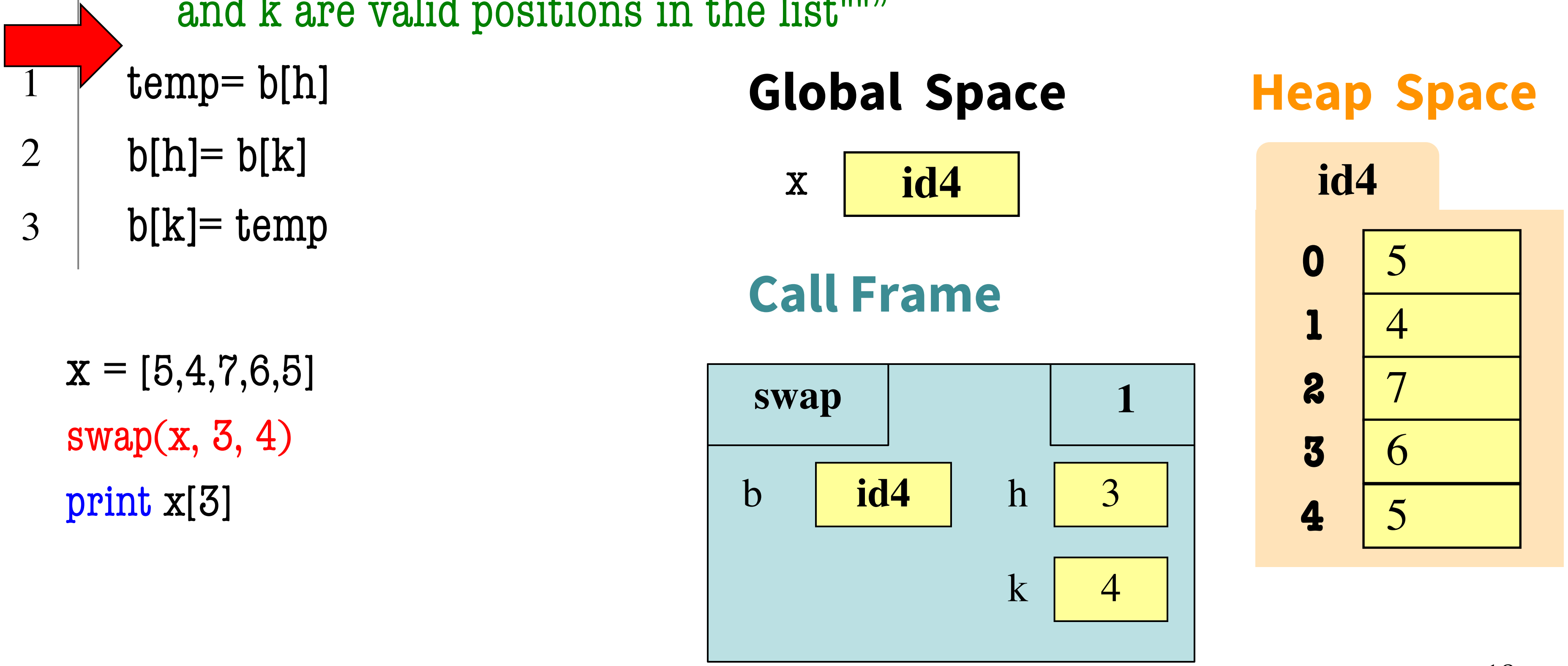
What gets printed?

A: 5 **CORRECT**
B: 6
C: Something else
D: I don't know

2nd Clicker Explanation (1)

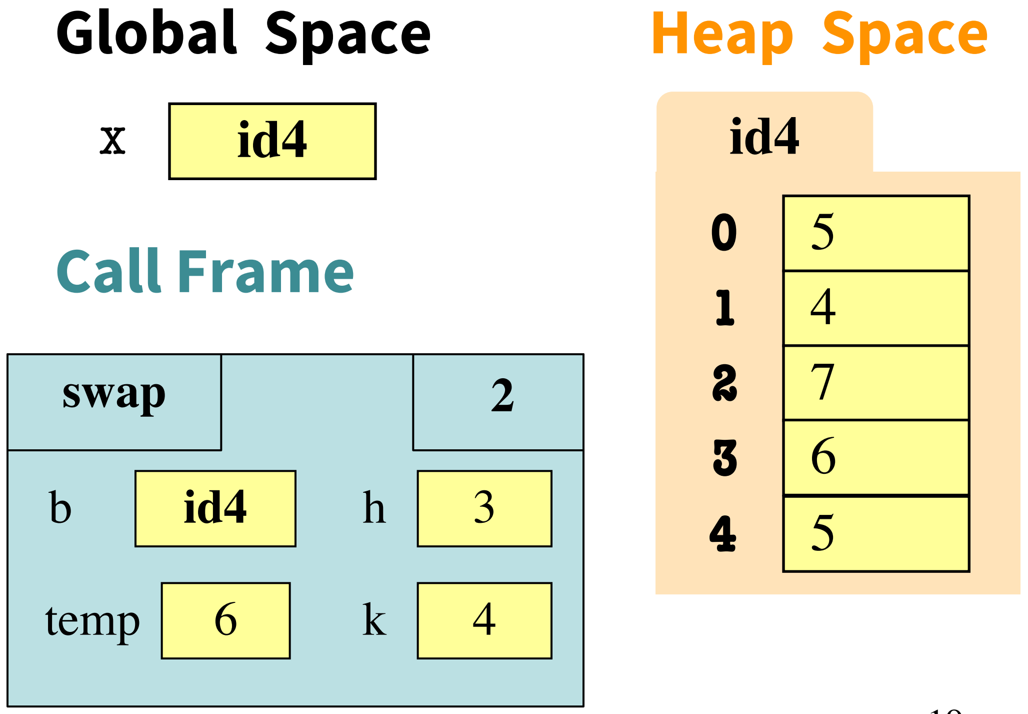
```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list"""
1   temp= b[h]
2   b[h]= b[k]
3   b[k]= temp
```

```
x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```



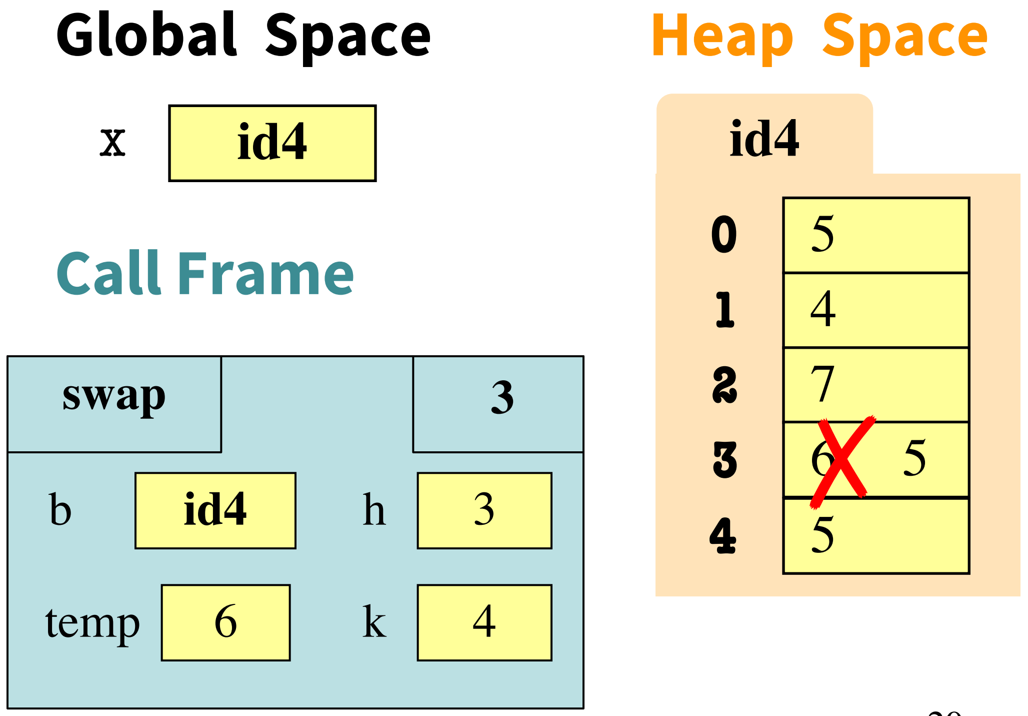
2nd Clicker Explanation (2)

```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list"""
1   temp= b[h]
2   b[h]= b[k]
3   b[k]= temp

x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```



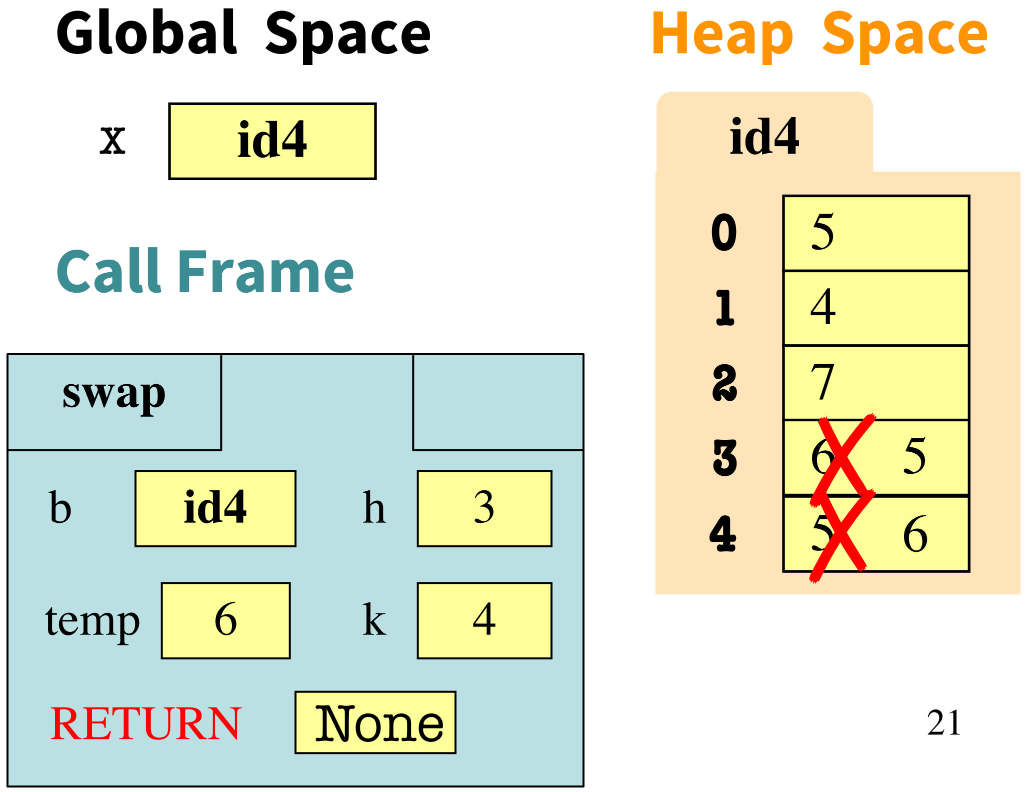
2nd Clicker Explanation (3)

```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list""”
    temp= b[h]
    b[h]= b[k]
    b[k]= temp

x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```



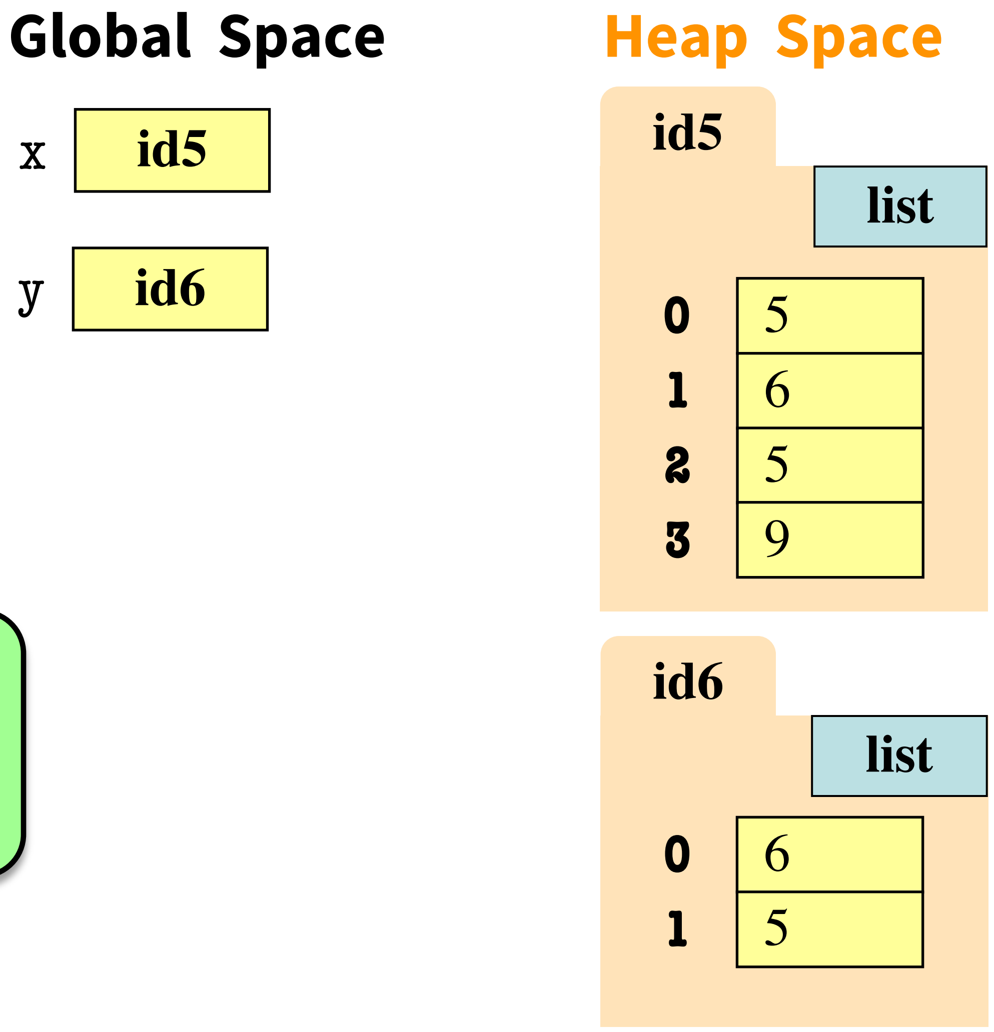
2nd Clicker Explanation (4)

```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list"""
    temp= b[h]
    b[h]= b[k]
    b[k]= temp

x = [5,4,7,6,5]
swap(x, 3, 4)
print x[3]
```



List Slices Make Copies

```
x = [5, 6, 5, 9]
y = x[1:3]
```



copy means
new folder

3rd Clicker Question

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> y = x[1:]
>>> y[0] = 7
```

- What is x[1]?

A: 7
B: 5
C: 6
D: **ERROR**
E: I don't know

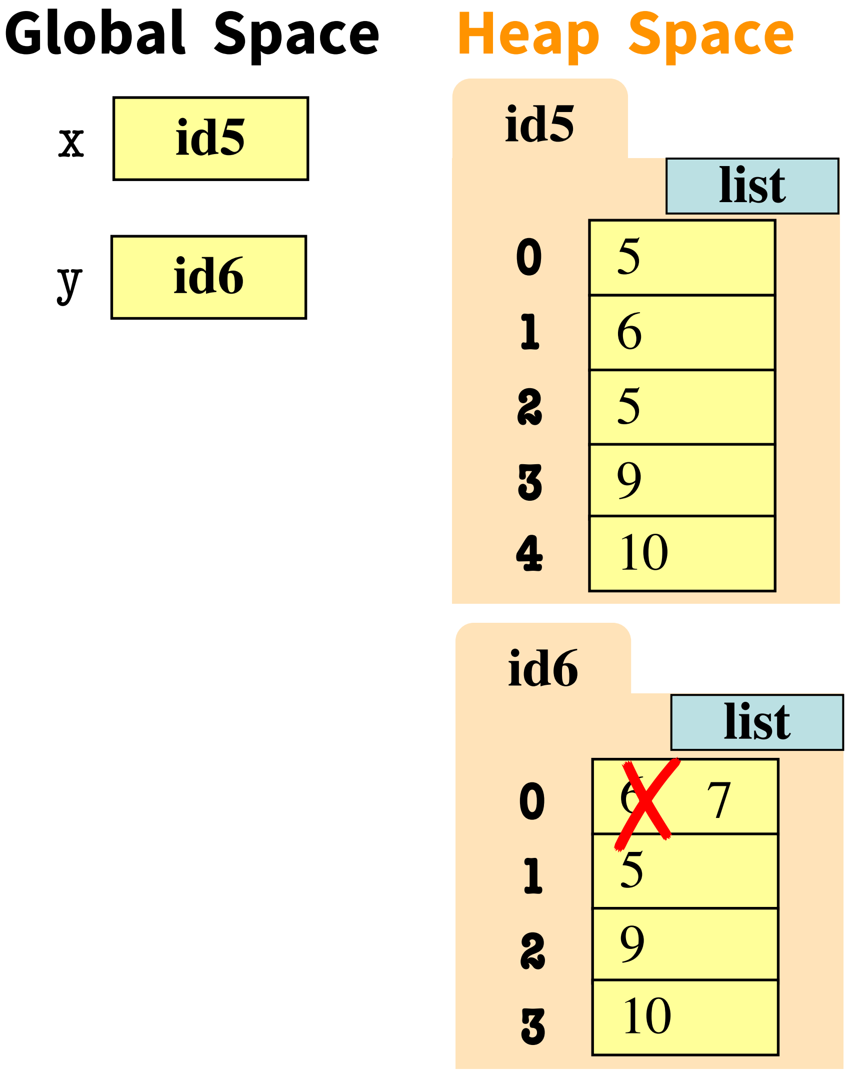
3rd Clicker Answer

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> y = x[1:]
>>> y[0] = 7
```

- What is x[1]?

A: 7
B: 5
C: 6 **CORRECT**
D: **ERROR**
E: I don't know



4th Clicker Question

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> y = x
>>> y[1] = 7
```

- What is `x[1]`?

A: 7
B: 5
C: 6
D: **ERROR**
E: I don’t know

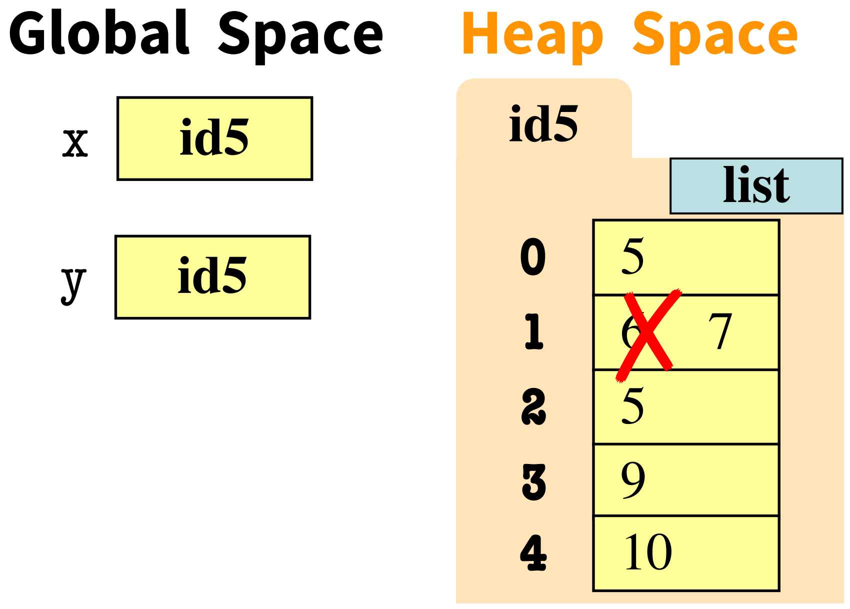
4th Clicker Answer

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> y = x
>>> y[1] = 7
```

- What is `x[1]`?

A: 7 **CORRECT**
B: 5
C: 6
D: **ERROR**
E: I don't know



Lists and Expressions / 5th Clicker Q

- List brackets [] can contain expressions
- This is a list **expression**
  - Python must evaluate it
  - Evaluates each expression
  - Puts the value in the list
- Example:

```
>>> a = [1+2,3+4,5+6]
>>> a
[3, 7, 11]
```

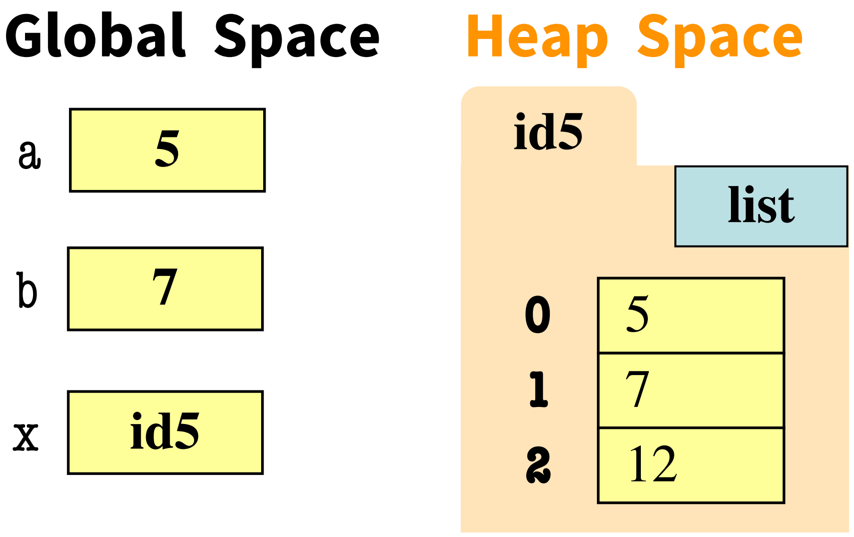
- Execute the following:

```
>>> a = 5
>>> b = 7
>>> x = [a, b, a+b]
```

- What is x[2]?

A: 'a+b'
B: 12
C: 57
D: **ERROR**
E: I don’t know

Lists and Expressions / 5th Clicker A



- Execute the following:

```
>>> a = 5
>>> b = 7
>>> x = [a, b, a+b]
```

- What is x[2]?

A: 'a+b'
B: 12 **CORRECT**
C: 57
D: **ERROR**
E: I don't know

Lists and Strings Go Hand in Hand

text.split(<sep>): return a list of words in text (separated by <sep>, or whitespace by default)

<sep>.join(words): concatenate the items in the list of strings words, separated by <sep>.

```
>>> text = 'A sentence is just\n a list of words'
>>> words = text.split()
>>> words
['A', 'sentence', 'is', 'just', 'a', 'list', 'of', 'words']
>>> lines = text.split('\n')
>>> lines
['A sentence is just', ' a list of words']
>>> hyphenated = '-'.join(words)
>>> hyphenated
'A-sentence-is-just-a-list-of-words'
>>> hyphenated2 = '-
'.join(lines[0].split()+lines[1].split())
>>> hyphenated2
'A-sentence-is-just-a-list-of-words'
```

Turns string into a list of words

Turns string into a list of lines

Combines elements with hyphens

Merges 2 lists, combines elements with hyphens