Python CS-521

Eugene Pinsky
Department of Computer Science
Metropolitan College, Boston University
Boston, MA 02215
email: epinsky@bu.edu

March 26, 2020

Abstract

This course will present an effective approach to help you learn Python. With extensive use of graphical illustrations, we will build understanding of Python and its capabilities by learning through many simple examples and analogies. The class will involve active student participation, discussions, and programming exercises. This approach will help you build a strong foundation in Python that you will be able to effectively apply in real-job situations and future courses.

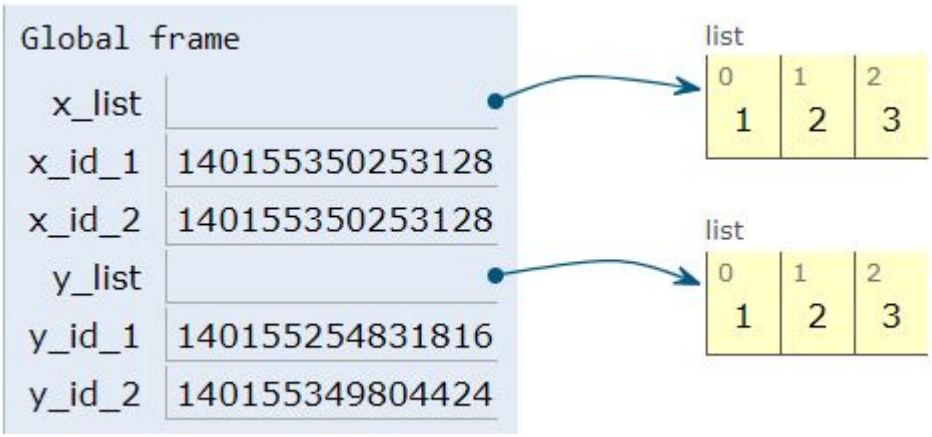
SORTING

Sorting in Python

- two built-in methods
  1. *sort*() - in-place for lists
  2. *sorted* - new sorted object
- *sorted* for any iterable object

Sort() vs. *Sorted*()

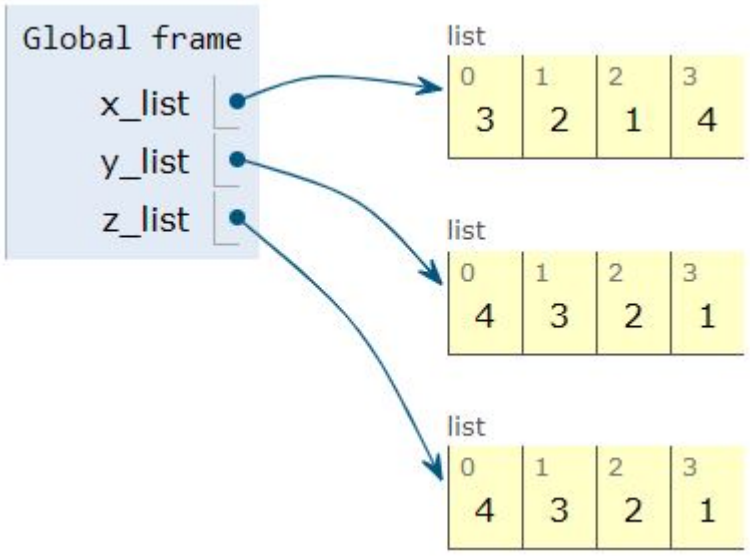
```
x_list = [3, 2, 1]
x_id_1 = id(x_list)
x_list.sort()
x_id_2 = id(x_list)

y_list = [3, 2, 1]
y_id_1 = id(y_list)
y_list = sorted(y_list)
y_id_2 = id(y_list)
```



Ascending/Descending

- optional parameter *reverse*

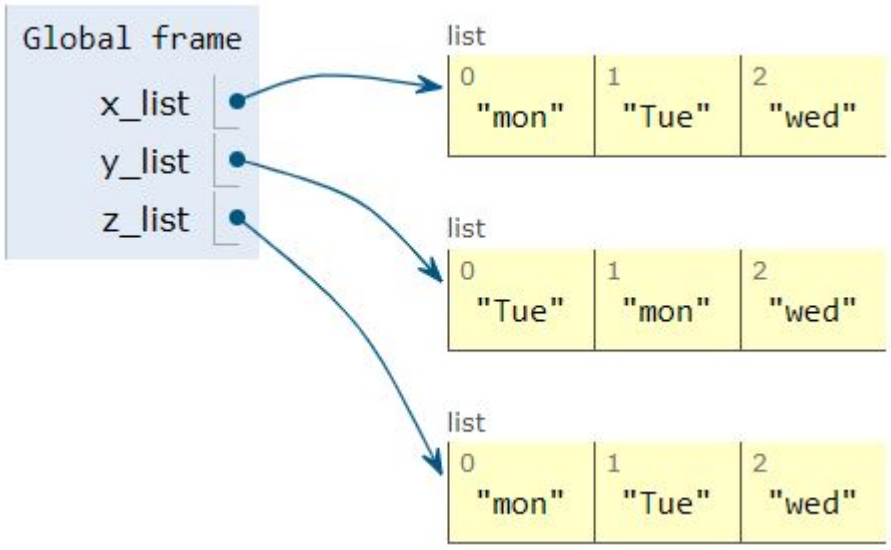
```
x_list = [3, 2, 1, 4]
y_list = [3, 2, 1, 4]
y_list.sort(reverse = True)

z_list = [3, 2, 1, 4]
z_list = sorted(z_list, reverse = True)
```



key Parameter

- specifies comparison function

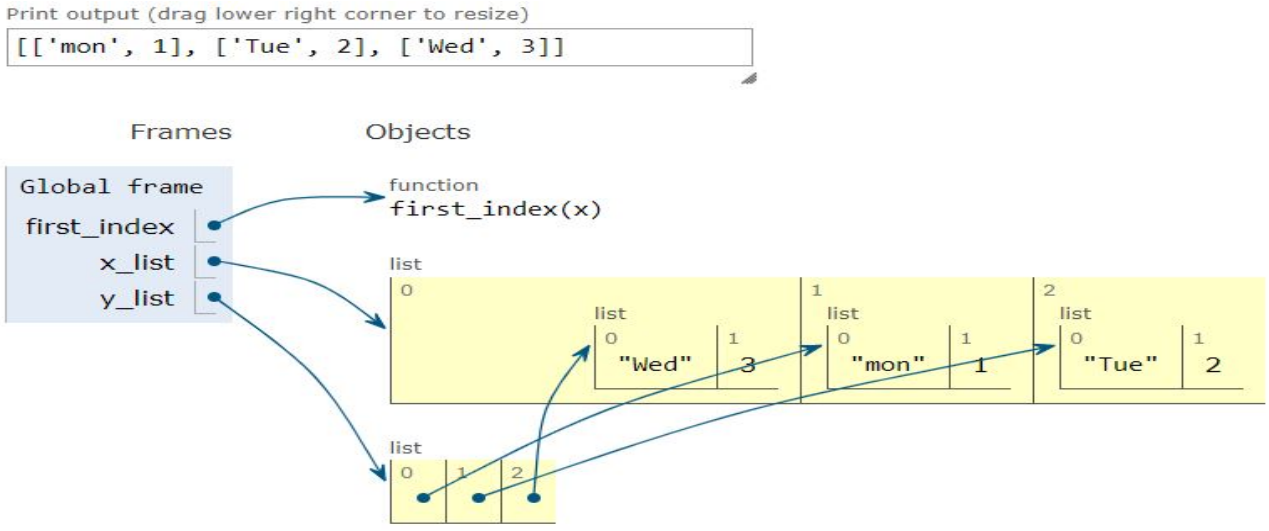
```
x_list = ['Wed', 'mon', 'Tue']

y_list = sorted(x_list)

# case insensitive sort
z_list = sorted(x_list, key = str.upper)
```



key Parameter

- can use object indices

```
def first_index(x):
    return x[1]

x_list = [['Wed',3], ['mon',1], ['Tue',2]]
y_list = sorted(x_list, key=first_index)
print(y_list)
```

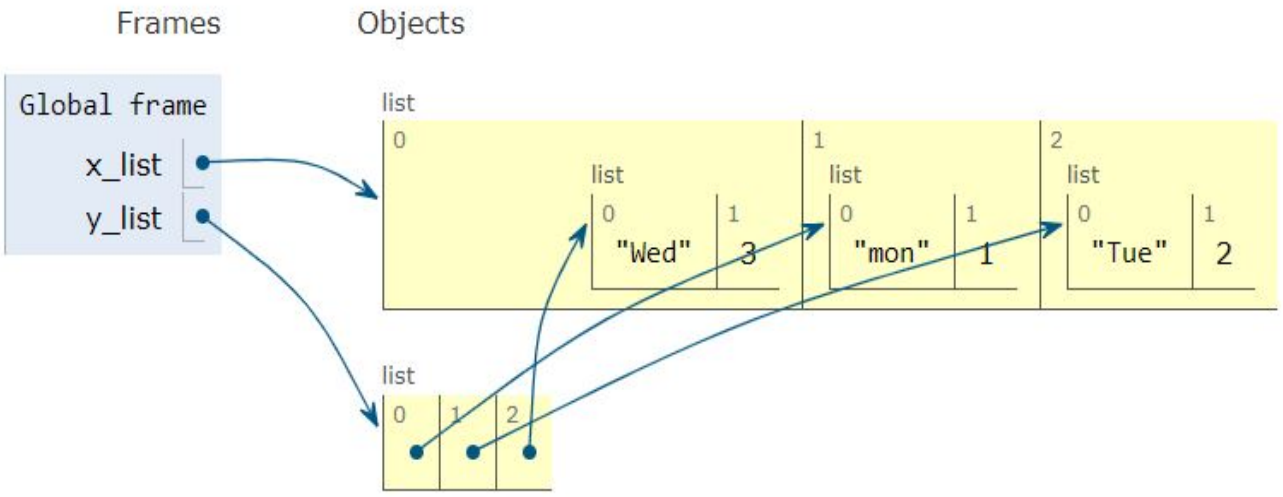


key Parameter (cont'd)

```
x_list = [['Wed',3], ['mon',1], ['Tue',2]]

y_list = sorted(x_list, key=lambda x: x[1])
print(y_list)
```

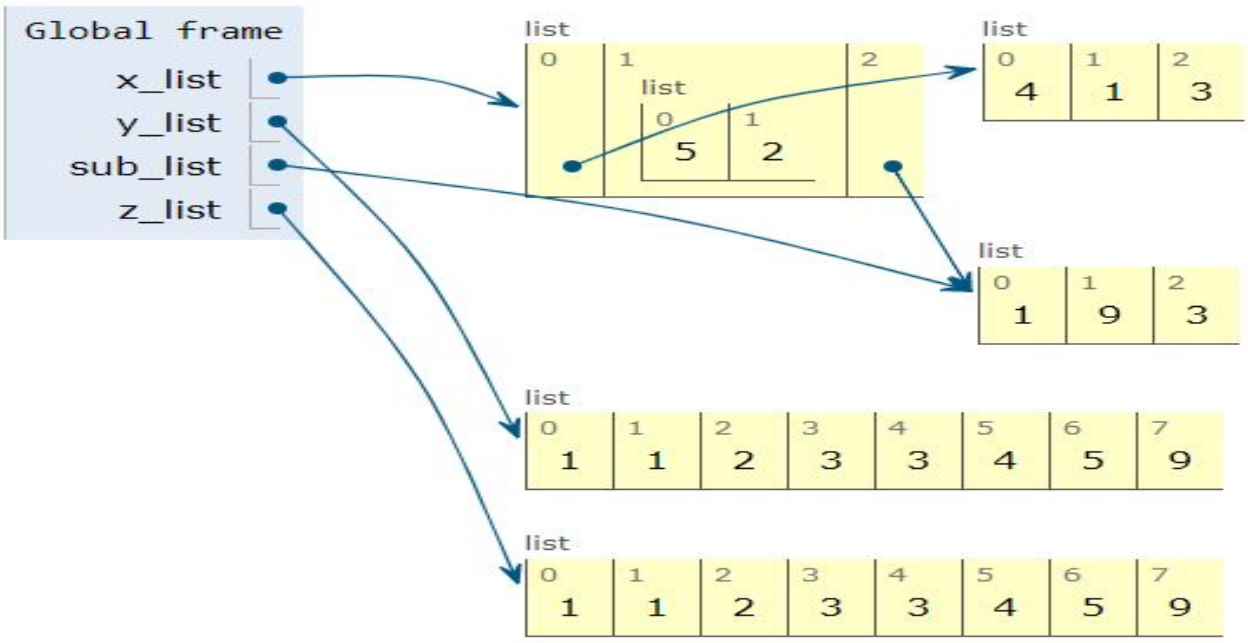
Print output (drag lower right corner to resize)

```
[['mon', 1], ['Tue', 2], ['Wed', 3]]
```



Sort Flatten List

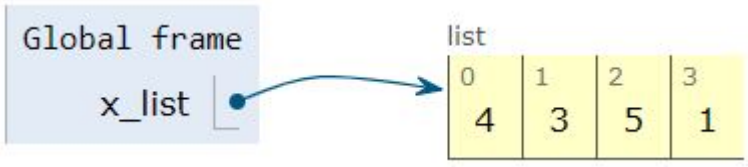
```
x_list = [[4, 1, 3],[5,2],[1,9,3]]

y_list = []
for sub_list in x_list:
    y_list.extend(sub_list)
y_list.sort()

z_list = sorted([j for i in x_list for j in i])
```



Bubble Sort

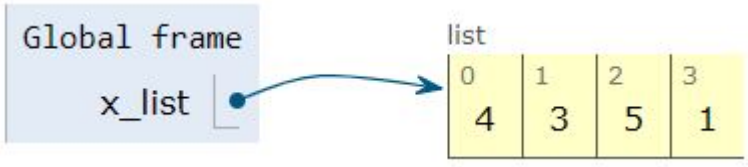
- given a list with n elements
- examine pairs of adjacent elements
- swap elements
- max elelemt is "bubbled" to the right
- repeat for $n - 1$ elements

Bubble Sort

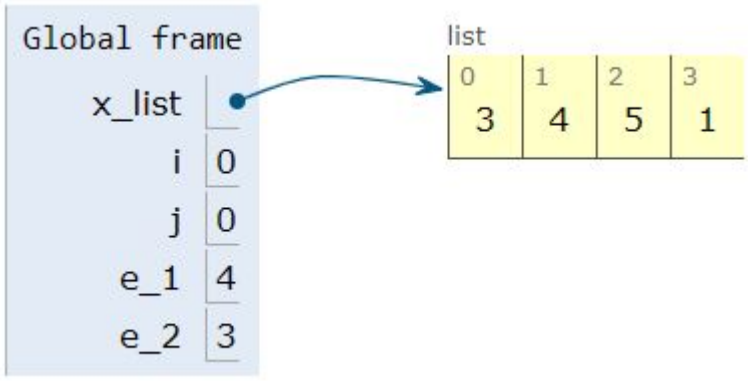



```
x_list = [4, 3, 5, 1]

for i in range(len(x_list)):
    for j in range(len(x_list) - 1):
        e_1 = x_list[j]
        e_2 = x_list[j+1]
        if e_1 > e_2:
            x_list[j]   = e_2
            x_list[j+1] = e_1
```

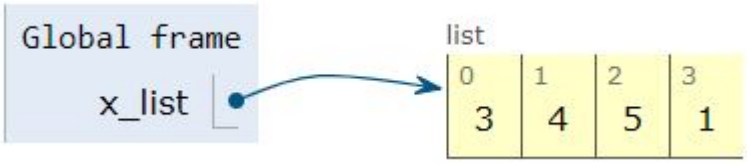
Bubble Sort (cont'd)



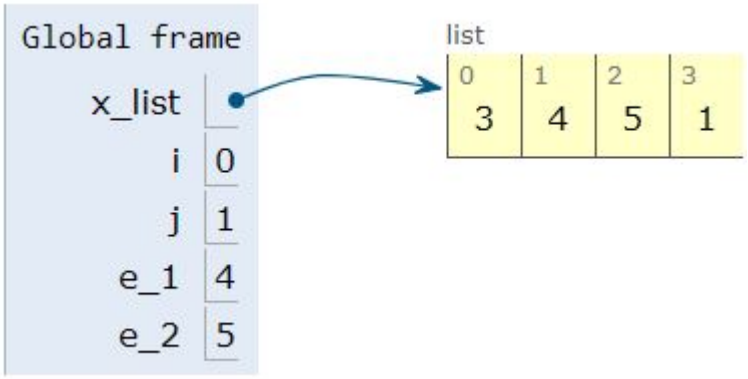
- swap 4 and 3



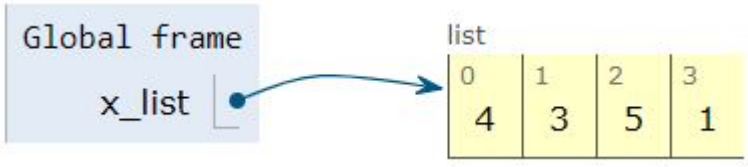
Bubble Sort (cont'd)



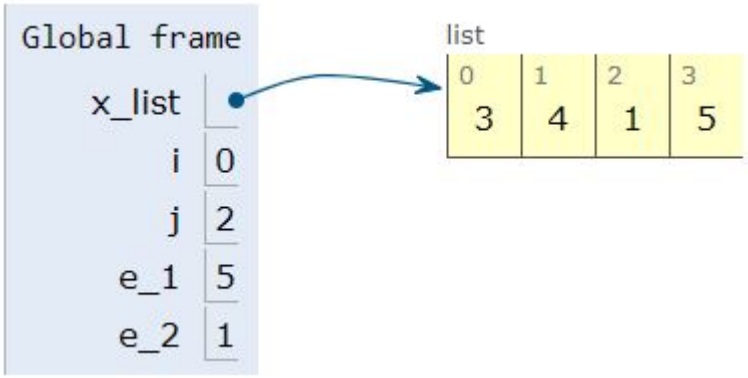
- do not swap 4 and 5



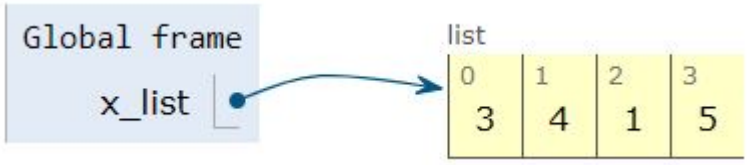
Bubble Sort (cont'd)



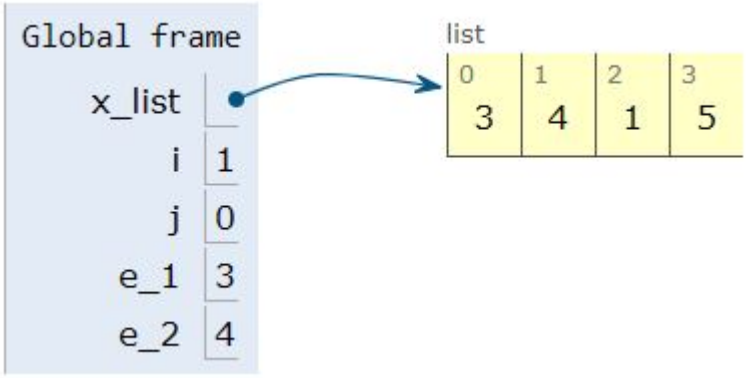
- swap 5 and 1



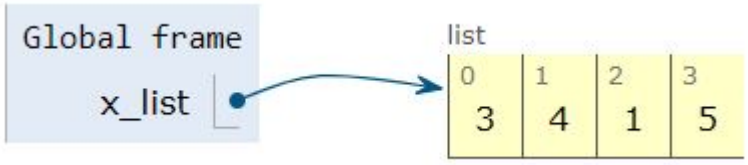
Bubble Sort (cont'd)



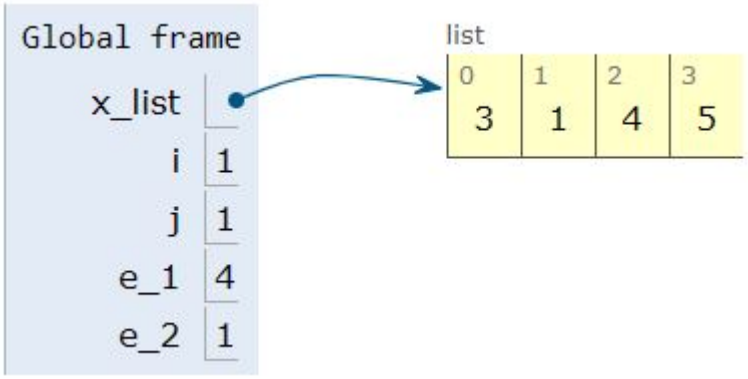
- do not swap 3 an 4



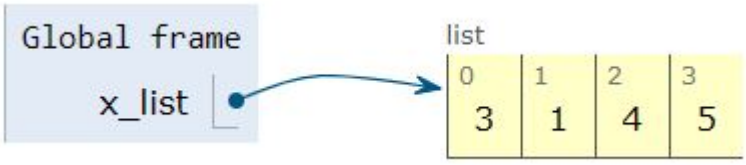
Bubble Sort (cont'd)



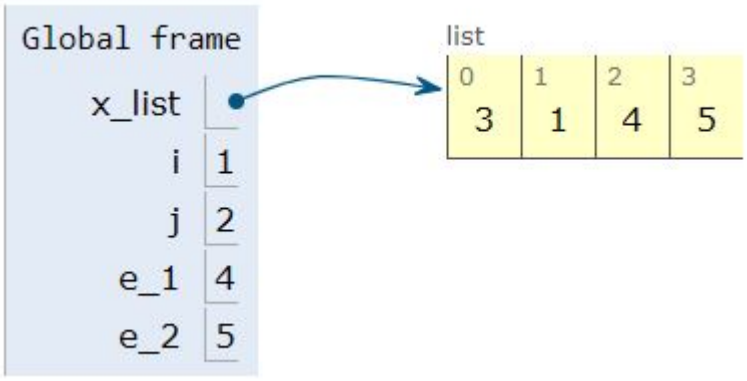
- swap 4 and 1



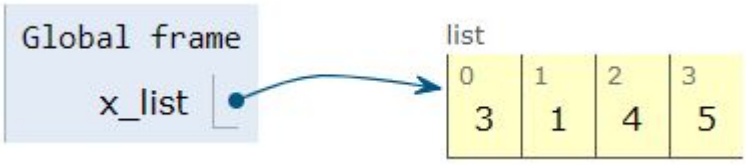
Bubble Sort (cont'd)



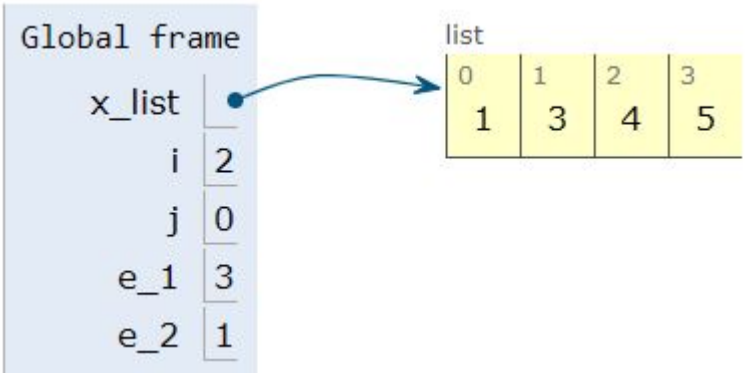
- do not swap 4 and 5



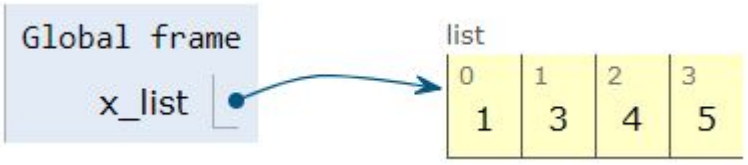
Bubble Sort (cont'd)



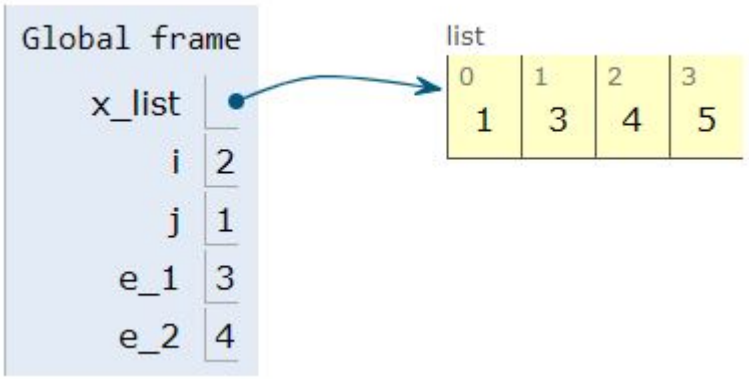
- swap 1 and 3



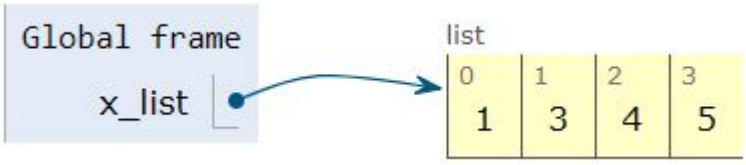
Bubble Sort (cont'd)



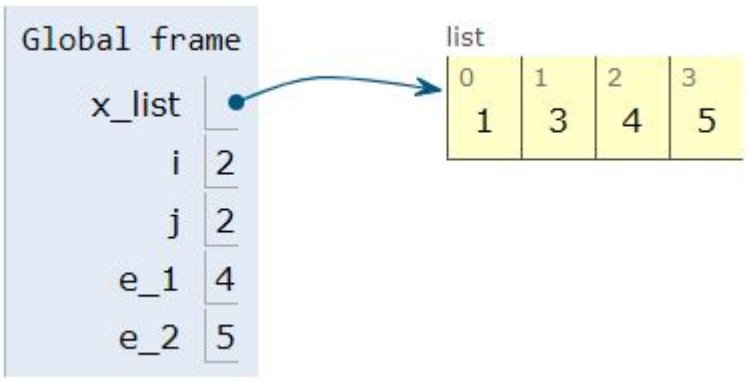
- do not swap 3 and 4



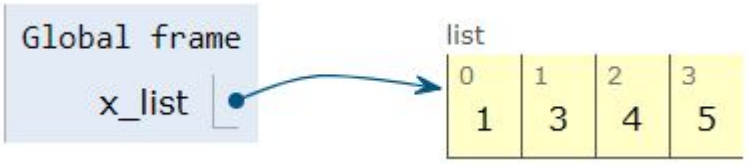
Bubble Sort (cont'd)



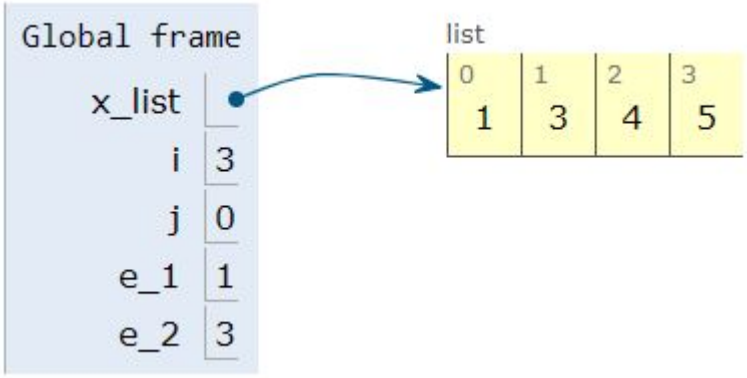
- do not swap 4 and 5



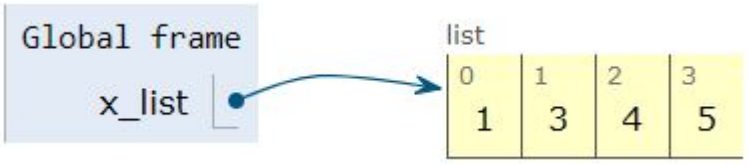
Bubble Sort (cont'd)



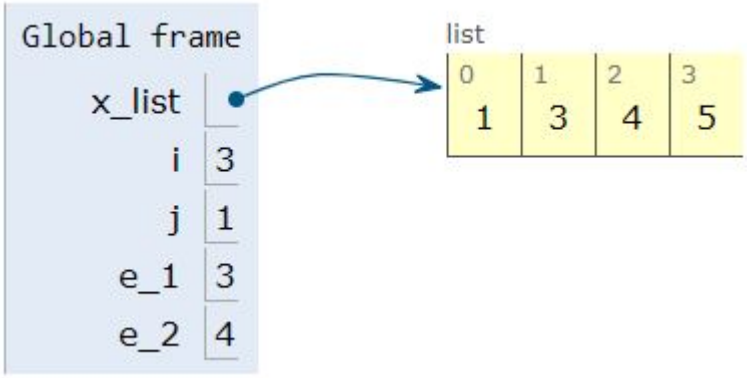
- do not swap 1 and 3



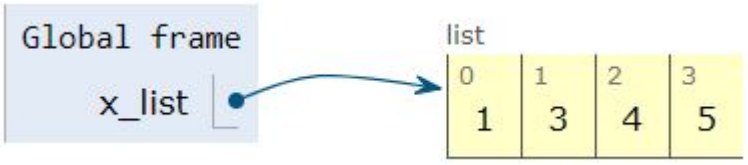
Bubble Sort (cont'd)



- do not swap 3 and 4



Bubble Sort (cont'd)



- do not swap 4 and 5