Here is Bubble Sort again:

In [1]:
```
def bubble_sort(L):
    for i in range(len(L)):
        #Set swapped to False. If nothing changes during a pass, we break.
        #If that happens, the runtime is better than the worst-case runtime
        #for a list of size n
        swapped = False

        #in the worst case, we need n-1 passes
        #The worst case happens when the smallest element is at L[-1], since
        #the smallest element only shifts one position to the left with each pass
        for j in range(len(L)-1-i):
            if L[j] > L[j+1]:
                L[j], L[j+1] = L[j+1], L[j]
                swapped = True
        if not swapped:
            break
```

Runtime Complexity Analysis -- Bubble Sort

Unlike with Selection Sort, Bubble Sort *can* terminate early -- if we `break` because a sweep didn't result in any two elements being swapped, the function returns faster.

We know that Bubble Sort will not run for more than `n` sweeps (where `n = len(L)`), just because the outer loop will not run for more than `n` iterations. (Recall that in the previous lecture we argued that Bubble Sort only ever needs at most `n-1` sweeps.

Is it ever the case that Bubble Sort needs all `n-1` sweeps to sort the list? Yes.

Note what happens when the smallest element of the list is initially the last element of the list:

In [2]:
```python
def bubble_sort(L):
    for i in range(len(L)):
        swapped = False
        for j in range(len(L)-1-i):
            if L[j] > L[j+1]:
                L[j], L[j+1] = L[j+1], L[j]
                print("Swapped", L[j], "and", L[j+1])
                print(L)
                swapped = True
            else:
                print("No need to swap", L[j], "and", L[j+1])
                print(L)
        if not swapped:
            return

        print("====================================")
if __name__ == '__main__':
    L = [2, 3, 4, 5, 1]
    bubble_sort(L)
```

```
No need to swap 2 and 3
[2, 3, 4, 5, 1]
No need to swap 3 and 4
[2, 3, 4, 5, 1]
No need to swap 4 and 5
[2, 3, 4, 5, 1]
Swapped 1 and 5
[2, 3, 4, 1, 5]
====================================
No need to swap 2 and 3
[2, 3, 4, 1, 5]
No need to swap 3 and 4
[2, 3, 4, 1, 5]
Swapped 1 and 4
[2, 3, 1, 4, 5]
====================================
No need to swap 2 and 3
[2, 3, 1, 4, 5]
Swapped 1 and 3
[2, 1, 3, 4, 5]
====================================
Swapped 1 and 2
[1, 2, 3, 4, 5]
====================================
```

Note that the `1` only moves left *once* per sweep. That makes sense: any element can only move left once during the sweep (but an element can move to the right many times.)

We can therefore conclude that the in the **worst case**, Bubble Sort does not return before performing all `n` iterations of the outer loop.

Let's now figure out the worst-case runtime complexity of Bubble Sort for a list of length `n` by counting how many times the inner block repeats.

At iteration `0`, the block runs for n-1-0 times At iteration `1`, the block runs for n-1-1 times At iteration `n-1`, the block runs for n-1-(n-1)=0 times

So in total, the block runs $\sum_{i=0}^{n-1}(n-i-1)$ times.

$\sum_{i=0}^{n-1}(n-i-1) = n^2 - \sum_{i=0}^{n-1} i - n = n^2 - n(n-1)/2 - n = n^2 - n^2/2 + n/2 - n = n^2/2 - n/2$

(To compute $\sum_{i=0}^{n-1} i$, we use the fact that $\sum_{j=1}^{m} j = m(m+1)/2$, and substitute $m = n-1$.)

We can therefore conclude that the worst-case runtime complexity of Bubble Sort if $\mathcal{O}(n^2)$, just like Selection Sort.