

# Bubble Sorting Algorithm

## Step 1 - Build a List

In python this is very easy, we will use the random library to generate a list of 10 numbers within a range of 1-100. The following code will do this

```
1 import random
2
3 itemsToSort=random.sample(range(100), 10)
4 print("Items (start)\t" + str(itemsToSort))
5
```

Line 1 tells Python that we wish to use the random library.

Line 3 generates the list and assigns it to the new variable **itemsToSort**

Line 4 prints out the randomised list so we can see it in the console.

Press Ctrl-Enter (if using [repl.it](https://repl.it)) and the following should be printed in the console window.

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start)    [59, 72, 32, 96, 92, 8, 28, 14, 35, 53]
=> None
>
```

## Step 2 - Write a Swap Function

This function will accept two indexes and it will swap the items in our **itemsToSort** list. Type the following code

The swap variable (created on line 7) is used to keep the item in **fromIndex** whilst the item in **toIndex** is being copied over it. This function operates on the **itemsToSort** variable declared above, we could have passed the list in as an argument too, which would make the function more re-usable.

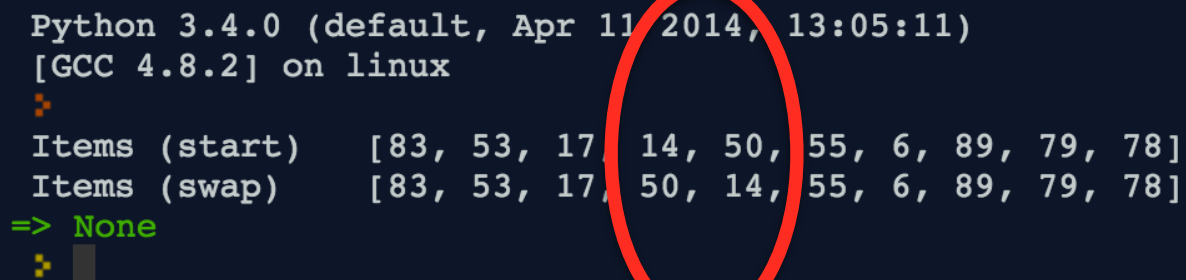
```
6 def swapItems(fromIndex, toIndex):
7     swap = itemsToSort[fromIndex]
8     itemsToSort[fromIndex] = itemsToSort[toIndex]
9     itemsToSort[toIndex] = swap
10
```

## Step 3 - Test the Swap Function

In order to test the function, first we must call it. At the bottom of the code type the following lines.

```
10
11 swapItems(3, 4);
12 print("Items (swap)\t" + str(itemsToSort))
```

Line 11 calls our new function, Line 12 prints the list after the swap. From this we can visually inspect the output to see if it has worked.



```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start)    [83, 53, 17, 14, 50, 55, 6, 89, 79, 78]
Items (swap)    [83, 53, 17, 50, 14, 55, 6, 89, 79, 78]
=> None
>
```

In order to be more systematic in testing our function, we will write a small test function. This test function will execute the swap function, and check that it works properly. Once we have written this test, we can always use it to demonstrate the code working, without even having to do visual inspection as we have above.

Replace the code we wrote before with the following

```
11 def testSwapItems():
12     item3before = itemsToSort[3]
13     item4before = itemsToSort[4]
14     swapItems(3, 4)
15     assert itemsToSort[4] == item3before, "item 4 has not been swapped"
16     assert itemsToSort[3] == item4before, "item 3 has not been swapped"
17     print("swapItems() Successfully passed test")
18
19 testSwapItems()
```

Let's break this down by line

- 12. The variable item3before remembers the value in index 3 *before* the swap is called.
- 13. This variable remembers the value in index 4.
- 14. Calls the swapItems function with indexes 3 and 4
- 15. Checks that the item in index 4 is equal to the item that used to be in index 3. It will print a message if this is not correct for any reason.
- 16. Checks the opposite item, also prints a message.
- 17. Prints to console so we can see it has run ok

Run this code and the following should be shown

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start)    [99, 92, 40, 46, 67, 10, 29, 31, 74, 56]
swapItems() Succesfully passed test
=> None
>
```

Assert statements are special statements that allow a programmer to state something that they expect at a given moment in time. In this case, at line 15, we expect that `itemsToSort[4]` contains the value that used to be in `itemsToSort[3]`. If this was not true, the program would throw an error.

To demonstrate this, comment out the line that calls the swap function. This will mean the swap is not run, so the test should fail.

```
11 def testSwapItems():
12     item3before = itemsToSort[3]
13     item4before = itemsToSort[4]
14     #swapItems(3, 4)
15     assert itemsToSort[4] == item3before, "item 4 has not been swapped"
16     assert itemsToSort[3] == item4before, "item 3 has not been swapped"
17     print("swapItems() Succesfully passed test")
18
```

Run the code again and the following should be shown

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start)    [59, 68, 32, 10, 42, 89, 2, 75, 76, 70]
Traceback (most recent call last):
  File "python", line 19, in <module>
  File "python", line 15, in testSwapItems
AssertionError: item 4 has not been swapped
>
```

The last line shows the message we wrote, which helps us (the programmer) find the error and fix it. Uncomment the line 14 and we will continue.

## Step 4 - Write the Comparison Function

The comparison function will be given two arguments, the `fromIndex` and `toIndex` (same as `swap`). But rather than simply swap the items, it first checks to see if they are in the correct order. If two items are already in the correct order, there is no need to swap them.

The code for the comparison looks like this

```
21 def compareItems(fromIndex, toIndex):  
22     if (itemsToSort[fromIndex] > itemsToSort[toIndex]):  
23         swapItems(fromIndex, toIndex)  
24
```

Line by line

21. This defines the function, it declares two arguments (`fromIndex` and `toIndex`).

22. This conditional statement compares the item in the `fromIndex` with the item in the `toIndex`. If the item in the lower position is actually larger than the item in the higher position, then the `swap` function is called.

Now we need to test the function!

## Step 5 - Test the Comparison Function

For this test to run properly, we must run two tests. The first will compare two items that we know are in the correct order, so they *should not* be swapped. The next will compare two items which we know are in the wrong order, so they *should* be swapped.

First write the code for the test function itself

```
25 def testCompareItems(fromIndex, toIndex, expectSwap):
26     itemFromBefore = itemsToSort[fromIndex]
27     itemToBefore = itemsToSort[toIndex]
28     compareItems(fromIndex, toIndex)
29     if (expectSwap):
30         assert itemsToSort[fromIndex] == itemToBefore, 'swap from is wrong'
31         assert itemsToSort[toIndex] == itemFromBefore, 'swap to is wrong'
32         print("compareItems() test - Items correctly swapped")
33     else:
34         assert itemsToSort[fromIndex] == itemFromBefore, 'noSwap from is wrong'
35         assert itemsToSort[toIndex] == itemToBefore, 'noSwap to is wrong'
36         print("compareItems() test - Items correctly left alone")
37
```

Line by line

- 25. This defines a test function that requires arguments, it means we can use this function for both cases. The two indexes are passed in along with an indication that they should be swapped, or not.
- 26 - 27. Remember value in fromIndex and toIndex before compareItems() is called
- 28. Calls compareItems, passing in the indexes under test
- 29. The assertions are different depending on if we expect a swap to happen.
- 29 - 31. Check that from and to were correctly swapped, print a success indicator
- 33. If expectSwap was False, we will check that the items have not moved
- 34 - 36. Check that the items are in their starting positions, print success indicator

This function now must be called in order to run any test. Before calling this function a bit of setup is required. We will put some fixed values into the list, in fixed positions. This allows the test to have predictable results.

```
38 itemsToSort[4] = 56
39 itemsToSort[5] = 66
40 testCompareItems(4, 5, False)
41
42 itemsToSort[7] = 31
43 itemsToSort[8] = 12
44 testCompareItems(7, 8, True)
45
```

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start) [9, 77, 5, 78, 4, 89, 98, 22, 95, 93]
swapItems() Successfully passed test
compareItems() test - Items correctly left alone
compareItems() test - Items correctly swapped
=> None
>
```

Lines 38 - 39 put fixed values into positions 4 and 5. They are in the correct order, so a swap is not expected.

Line 40 then calls the function with these two indexes and indicates that a swap is not expected.

Lines 42 - 43 put fixed values into position 7 and 8. These are in the wrong order, so a swap will be required to get them into the correct order.

Line 44 calls the test function, indicating that a swap is expected.

When you run the code now, the output shown just above indicates that the tests have run correctly. Again, try changing the expectSwap value on lines 40 or 44 and see if you get the correct errors.

## Step 6 - Write the Sort Function

The sort function will use the comparison function in a repeated fashion until the list is in order.

The sort function itself looks like this.

```
47 def sortItems():
48     for top in reversed(range(1, len(itemsToSort))):
49         for current in range(0, top):
50             compareItems(current, current + 1)
51
```

Line by line

46. Defines the `sortItems()` function. This requires no arguments as it will operate on the whole of the `itemsToSort` list that we have already created.

47. This is quite complex, so let's break it up.

- The **top** variable indicates the last item in the list that has not yet been sorted. At the start of the algorithm, the last item in the list is the top. But each run through will bring the highest value to the top, so therefore **top** reduces by 1 each time
- We need to count down from the length of the `itemsToSort` to 1. The range function will then give us a list of numbers like this [1, 2, 3, 4, 5, 6, 7, 8, 9]
- The `reversed()` function takes this list and reverses it. This means that top will now count down from 9 to 0. It will now look like this [9, 8, 7, 6, 5, 4, 3, 2, 1]

48. Within each iteration, we must count up from the first item to the last unsorted item. The statement inside the loop will then compare the **current** item with the **current + 1** item.

49. Call the `compareItems()` function itself. Each iteration will compare the **current** item with the one after it, if they are in the wrong order, they will be swapped.

The following code can be written on the end to call our function and reprint the list so we can see that it has been sorted.

```
50
51 sortItems();
52 print("Items (end)\t" + str(itemsToSort))
53
```

Run this code now, if everything has gone well, we should see something like this.

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start)    [10, 68, 46, 97, 3, 54, 87, 48, 30, 45]
swapItems() Successfully passed test
compareItems() test - Items correctly left alone
compareItems() test - Items correctly swapped
Items (end)      [3, 10, 12, 46, 56, 66, 68, 87, 31, 45]
=> None
>
```

## Step 7 - Test the Sort Function

We can visually inspect the printed list and check it is in order, but just like the other functions, it is a good idea to write a test. This test will do a single run through the items checking they are all in the correct order.

The test function looks like this.

```
54 def testSortItems():
55     random.shuffle(itemsToSort)
56     sortItems();
57     for x in range(0, len(itemsToSort) - 1):
58         assert itemsToSort[x + 1] >= itemsToSort[x], "Items in wrong order"
59     print("sortItems() test - Items in correct order")
60
```

54. Declares the test function, no arguments required

55. In order to run a valid test, we should shuffle the list at the start.

56. Call sortItems() to get the items back in order

57. Run through all the items and check they are in the right order. The loop goes up to the length -1. The minus 1 is required because the comparison will check x against x + 1, and we don't want to run off the end.

58. Assert that the current item is in the correct order relative to the next one in the list.

59. Print success

Call the testSortItems() by simply calling it.

```
60
61 testSortItems()
62
```

Run the code and the following should be shown. This indicates that the program is working correctly.

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
>
Items (start)    [94, 66, 13, 36, 69, 7, 39, 49, 41, 22]
swapItems() Successfully passed test
compareItems() test - Items correctly left alone
compareItems() test - Items correctly swapped
Items (end)      [12, 13, 22, 31, 39, 56, 66, 66, 69, 94]
sortItems() test - Items in correct order
=> None
>
```

## Step 8 - Tidy up and recap

Before we finish, lets rearrange the code a bit. The order will now be as follows

1. Functions required by sort
  1. swapItems()
  2. compareItems()
  3. sortItems()
2. Test functions
  1. testSwapItems()
  2. testCompareItems()
  3. testSortItems()
3. Call test functions
4. Demonstration call

The next page shows the reorganised code, when run the output should now look like this.

A working solution can be found here <https://repl.it/B2GZ/54>

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
❖
Items (start)    [53, 85, 3, 70, 28, 13, 27, 26, 40, 48]
swapItems() Succesfully passed test
compareItems() test - Items correctly left alone
compareItems() test - Items correctly swapped
sortItems() test - Items in correct order
Items (end)      [3, 12, 27, 28, 31, 48, 53, 56, 66, 85]
=> None
❖ █
```

**CONGRATULATIONS! YOU HAVE NOW IMPLEMENTED A BUBBLE SORT IN PYTHON**



```

1 import random
2
3 itemsToSort=random.sample(range(100), 10)
4 print("Items (start)\t" + str(itemsToSort))
5
6 def swapItems(fromIndex, toIndex):
7     swap = itemsToSort[fromIndex]
8     itemsToSort[fromIndex] = itemsToSort[toIndex]
9     itemsToSort[toIndex] = swap
10
11 def compareItems(fromIndex, toIndex):
12     if (itemsToSort[fromIndex] > itemsToSort[toIndex]):
13         swapItems(fromIndex, toIndex)
14
15 def sortItems():
16     for top in reversed(range(1, len(itemsToSort))):
17         for current in range(0, top):
18             compareItems(current, current + 1)
19
20 def testSwapItems():
21     item3before = itemsToSort[3]
22     item4before = itemsToSort[4]
23     swapItems(3, 4)
24     assert itemsToSort[4] == item3before, "item 4 has not been swapped"
25     assert itemsToSort[3] == item4before, "item 3 has not been swapped"
26     print("swapItems() Successfully passed test")
27
28 def testCompareItems(fromIndex, toIndex, expectSwap):
29     itemFromBefore = itemsToSort[fromIndex]
30     itemToBefore = itemsToSort[toIndex]
31     compareItems(fromIndex, toIndex)
32     if (expectSwap):
33         assert itemsToSort[fromIndex] == itemToBefore, 'swap from is wrong'
34         assert itemsToSort[toIndex] == itemFromBefore, 'swap to is wrong'
35         print("compareItems() test - Items correctly swapped")
36     else:
37         assert itemsToSort[fromIndex] == itemFromBefore, 'noSwap from is wrong'
38         assert itemsToSort[toIndex] == itemToBefore, 'noSwap to is wrong'
39         print("compareItems() test - Items correctly left alone")
40
41 def testSortItems():
42     random.shuffle(itemsToSort)
43     sortItems();
44     for x in range(0, len(itemsToSort) - 1):
45         assert itemsToSort[x + 1] >= itemsToSort[x], "Items in wrong order"
46     print("sortItems() test - Items in correct order")
47
48 testSwapItems()
49
50 itemsToSort[4] = 56
51 itemsToSort[5] = 66
52 testCompareItems(4, 5, False)
53
54 itemsToSort[7] = 31
55 itemsToSort[8] = 12
56 testCompareItems(7, 8, True)
57
58 testSortItems()
59
60 sortItems();
61 print("Items (end)\t" + str(itemsToSort))
62

```