Sorting Lists

We've considered how you might perform sorting operations in an earlier set of algorithms resources. We'll now have a look at how these are actually constructed in a computer program.

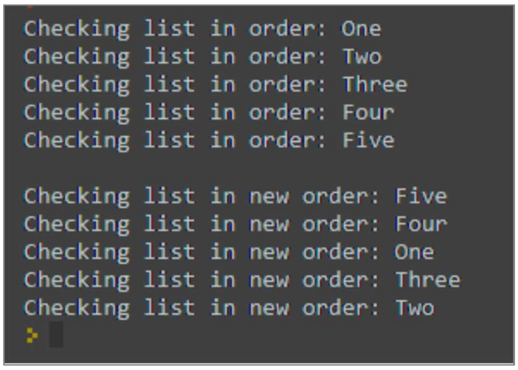
Aim: *To look at different methods of sorting items in a list.*

Task 1 – Sorting with the Built-In Function

Sorting lists in Python is as easy as using the *sort* function:

```
words_list.sort() #where words_list is the name of the list
```

Write a program saved as '**14.1 Sort**' that creates a list as shown on the right, displays it, sorts it alphabetically using the *sort* function and then displays it again in the new order.



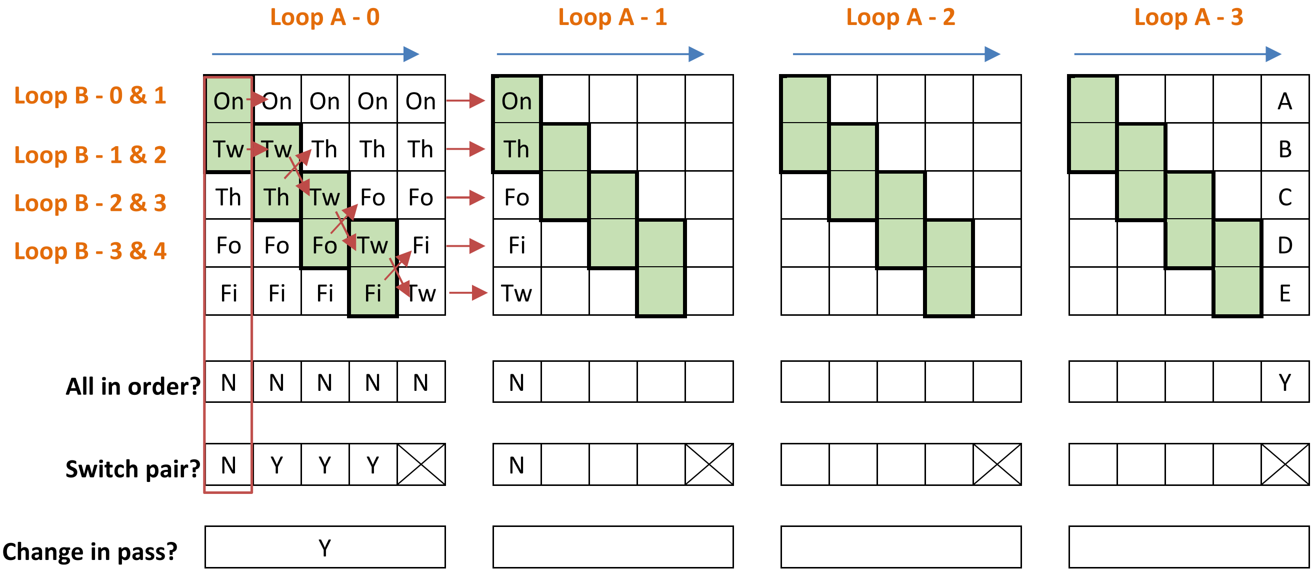
```
Checking list in order: One
Checking list in order: Two
Checking list in order: Three
Checking list in order: Four
Checking list in order: Five

Checking list in new order: Five
Checking list in new order: Four
Checking list in new order: One
Checking list in new order: Three
Checking list in new order: Two
>
```

Task 2 – Bubble Sort

Sort is a built-in function which runs code that's hidden away in the application. We're now going to look at some of the methods the *sort* function could be using behind the scenes to sort our data. The first is the bubble sort.

A very basic bubble sort algorithm inspects data one pair at a time, switching the pair if necessary. It compares data items 1 and 2, then 2 and 3, then 3 and 4 etc. When it gets to the end of the data, it starts again on another 'pass'. When a pass through the data results in no change then the data must be sorted.

a. Complete the table below showing what happens to the 5 items of data On, Tw, Th, Fo, Fi in a bubble sort. You should only compare the pairs of values in the green boxes, switching them if necessary while copying the data to the next column. Any data in white boxes remains the same in the next column.



Loop A - 0

Loop B - 0 & 1	On	On	On	On	On
Loop B - 1 & 2	Tw	Tw	Th	Th	Th
Loop B - 2 & 3	Th	Th	Tw	Fo	Fo
Loop B - 3 & 4	Fo	Fo	Fo	Tw	Fi
	Fi	Fi	Fi	Fi	Tw
All in order?	N	N	N	N	N
Switch pair?	N	Y	Y	Y	☒
Change in pass?	Y				

Loop A - 1

	On				
	Th				
	Fo				
	Fi				
	Tw				
All in order?	N				
Switch pair?	N				☒
Change in pass?					

Loop A - 2

All in order?					
Switch pair?					☒
Change in pass?					

Loop A - 3

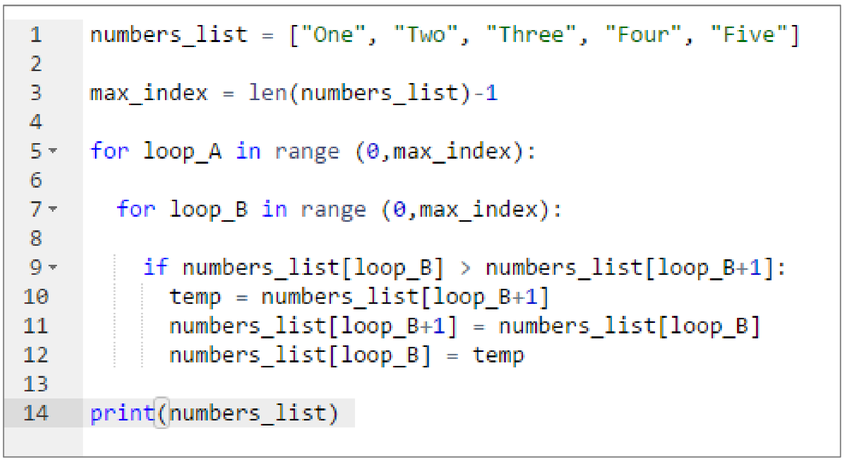
					A
					B
					C
					D
					E
All in order?					Y
Switch pair?					☒
Change in pass?					

All in order? *Place a 'Y' in this box if all items are sorted, else enter an 'N'.*
Switch pair? *Place a 'Y' in this box if the highlighted pair need to be switched, else enter an 'N'.*
Change in pass? *Place a 'Y' in this box if there are any switches in the whole pass, else enter an 'N'.*

Sorting Lists (page 2)

b. The program on the right recreates the Bubble sort from the last task. Set this up in repl.it and make sure that it is working correctly. Save as '**14.2 Bubble Sort A**'.

Note: *You have to make sure that you are consistent with your indentation. We have used 0, 1, 2 or 3 tabs at the start of each line. If you get indentation errors, it is often best to recreate them from scratch.*



```
1   numbers_list = ["One", "Two", "Three", "Four", "Five"]
2
3   max_index = len(numbers_list)-1
4
5   for loop_A in range (0,max_index):
6
7     for loop_B in range (0,max_index):
8
9       if numbers_list[loop_B] > numbers_list[loop_B+1]:
10        temp = numbers_list[loop_B+1]
11        numbers_list[loop_B+1] = numbers_list[loop_B]
12        numbers_list[loop_B] = temp
13
14  print(numbers_list)
```

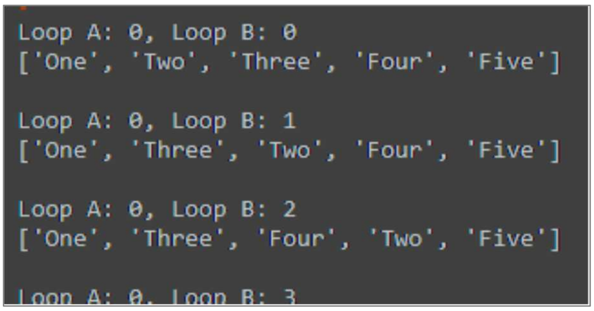
c. Explain with comments in your program what lines 9 to 12 do.

d. Draw a table showing how the values of *loop_A* and *loop_B* change as the program is run (0 0, 0 1, 0 2, 0 3, 1 0 etc.).

e. Indent line 14 so that it is inside the two loops (the same indentation as the *if* clause on line 9). You should now see a display of the full list at the end of each loop.

f. Add a spacer line so that each loop's output is separated.

g. Display the value of *loop_A* and *loop_B* before each pair is checked (as right).



```
Loop A: 0, Loop B: 0
['One', 'Two', 'Three', 'Four', 'Five']

Loop A: 0, Loop B: 1
['One', 'Three', 'Two', 'Four', 'Five']

Loop A: 0, Loop B: 2
['One', 'Three', 'Four', 'Two', 'Five']

Loop A: 0, Loop B: 3
```

h. Check your console readout against your answers to *Task 2a.*

Note: *The program doesn't display the first column in each group of 5 from the table on the previous page; you can add this if you like.*

i. Add further lines of code to display any other information in the console that will help analyse what is happening. For example, you could display the values being compared and say whether they are switched or not.

j. What is the <u>maximum</u> number of pair comparisons that might be required to sort a list of 5 items? What about 1000 items? Or 1 million?

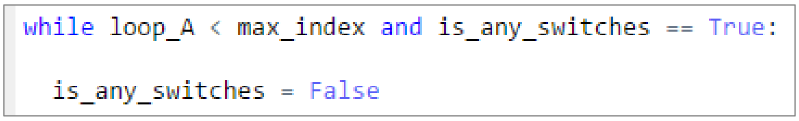
Extension

If no switches are made in an entire pass, then you do not need to complete any more passes. This can make a huge difference to the number of pair comparisons required to sort the data. In a copy of your program saved as '**14.2 Bubble Sort B**', try and add code to halt the process if a pass results in no switches. Here are some ideas:

- You could introduce a Boolean variable that is set to *False* at the start of each loop A and changes to *True* only when a switch is made. Note that the Boolean values *True* and *False* are not placed in quotes.

```
is_any_switches = True
```

- Depending on your solution, you may need to replace one of the 'For...Next' loops with a condition-controlled loop like the one shown.



```
while loop_A < max_index and is_any_switches == True:

  is_any_switches = False
```

Note: *The double equals sign is a comparison operator meaning 'is equal to'.*

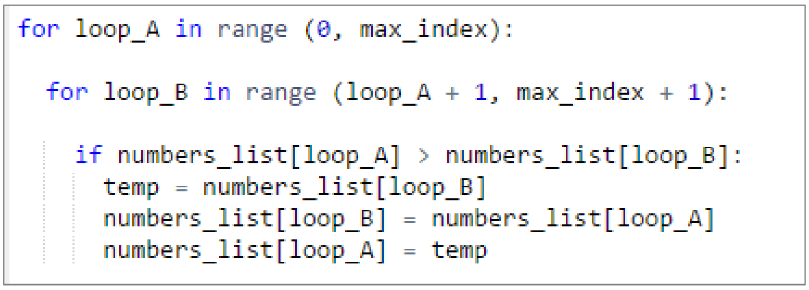
- Try creating the list nearly in order so that you can check that the process is halted quickly (e.g. Five, Four, One, Two, Three).
- Other changes will be required. Keep trying things until your solution works.

Sorting Lists (page 3)

Task 3 – A Better Bubble Sort?

The code below is part of a slightly different bubble sort. Starting with your solution to the last task, create this bubble sort and save as ‘**14.3 Better Bubble**’.

Complete a full analysis of this sorting algorithm using the methods from the previous tasks. Your collection of evidence about the efficiency of the algorithm might include:



```
for loop_A in range (0, max_index):

  for loop_B in range (loop_A + 1, max_index + 1):

    if numbers_list[loop_A] > numbers_list[loop_B]:
      temp = numbers_list[loop_B]
      numbers_list[loop_B] = numbers_list[loop_A]
      numbers_list[loop_A] = temp
```

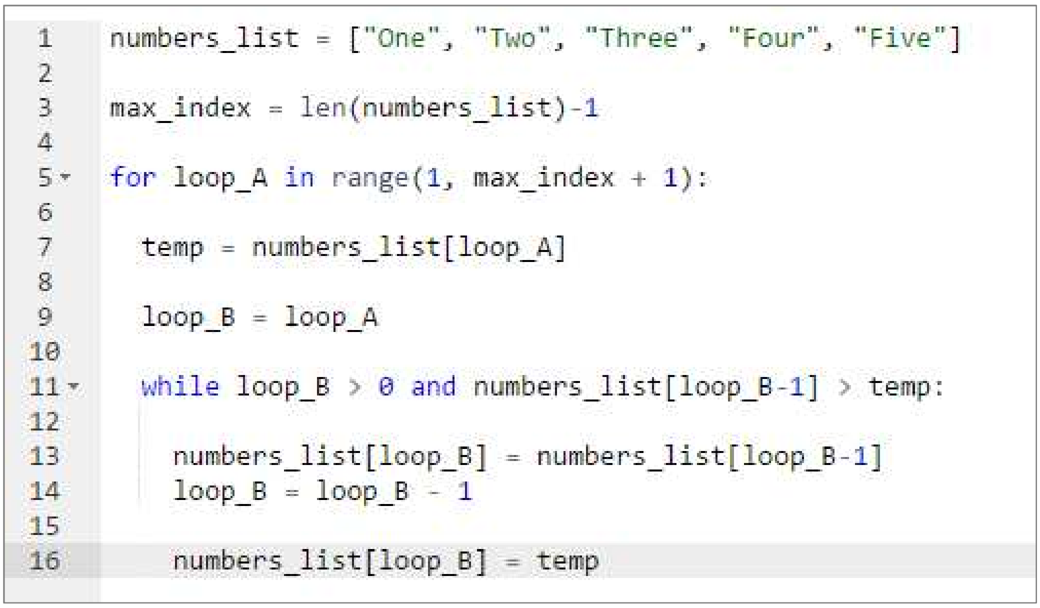
- Tables showing how the loop numbers change and the pair values are compared.
- Output to the console which traces the changing variables.
- The calculation of the maximum number of pair comparisons required for this algorithm to complete the sorting process.
- Taking another look at our first bubble sort and working out where the inefficiencies creep in. Think about the last value after the first pass and the last two values in the second pass etc.

Task 4 – Other Sorting Methods

Test the following sorting algorithms, saving each program with the name suggested in brackets.

Attempt to analyse the efficiency of each algorithm. You may have to be creative in the way that you add output to the console in order to track what is happening.

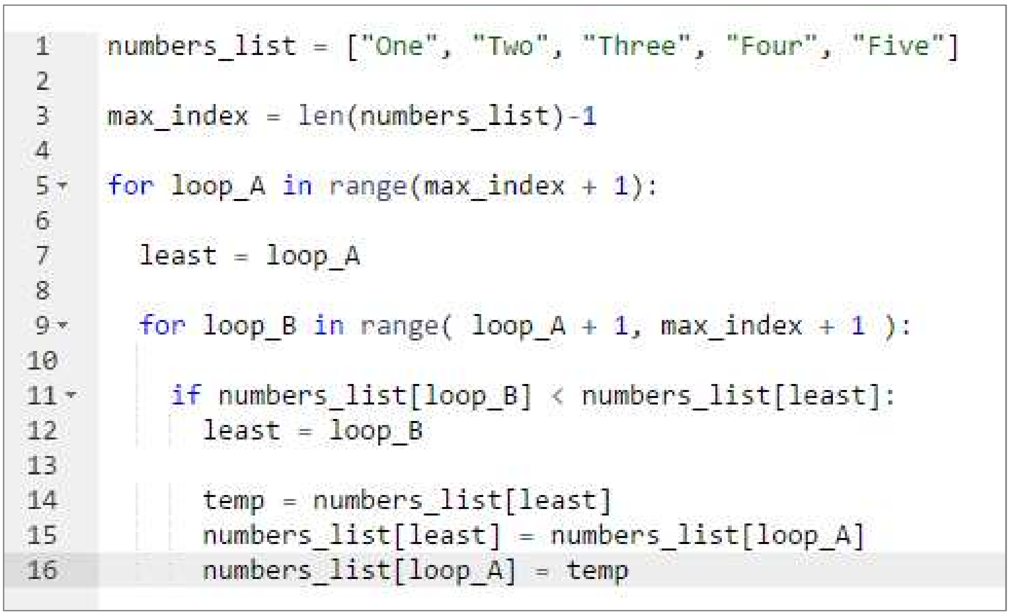
A. Insertion Sort (14.4A Insertion Sort)



```
numbers_list = ["One", "Two", "Three", "Four", "Five"]

max_index = len(numbers_list)-1

for loop_A in range(1, max_index + 1):

  temp = numbers_list[loop_A]

  loop_B = loop_A

  while loop_B > 0 and numbers_list[loop_B-1] > temp:

    numbers_list[loop_B] = numbers_list[loop_B-1]
    loop_B = loop_B - 1

    numbers_list[loop_B] = temp
```

Sorting Lists (page 4)

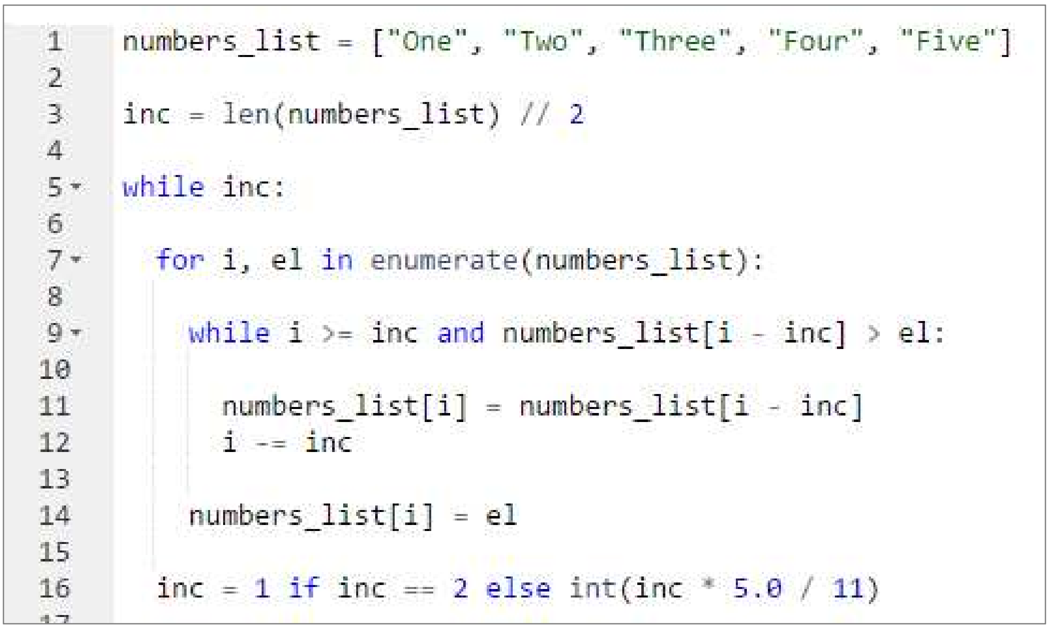
B. Selection Sort (14.4B Selection Sort)



```python
numbers_list = ["One", "Two", "Three", "Four", "Five"]

max_index = len(numbers_list)-1

for loop_A in range(max_index + 1):

  least = loop_A

  for loop_B in range( loop_A + 1, max_index + 1 ):

    if numbers_list[loop_B] < numbers_list[least]:
      least = loop_B

      temp = numbers_list[least]
      numbers_list[least] = numbers_list[loop_A]
      numbers_list[loop_A] = temp
```

Extension

C. Shell Sort (14.4C Shell Sort)



```python
numbers_list = ["One", "Two", "Three", "Four", "Five"]

inc = len(numbers_list) // 2

while inc:

  for i, el in enumerate(numbers_list):

    while i >= inc and numbers_list[i - inc] > el:

      numbers_list[i] = numbers_list[i - inc]
      i -= inc

    numbers_list[i] = el

  inc = 1 if inc == 2 else int(inc * 5.0 / 11)
```

Note: *You can find a range of programs for all these sorting methods online. They are subtly different to the ones shown here, but the general principles are the same.*

We will look at the more complex *Merge Sort* in a later task.