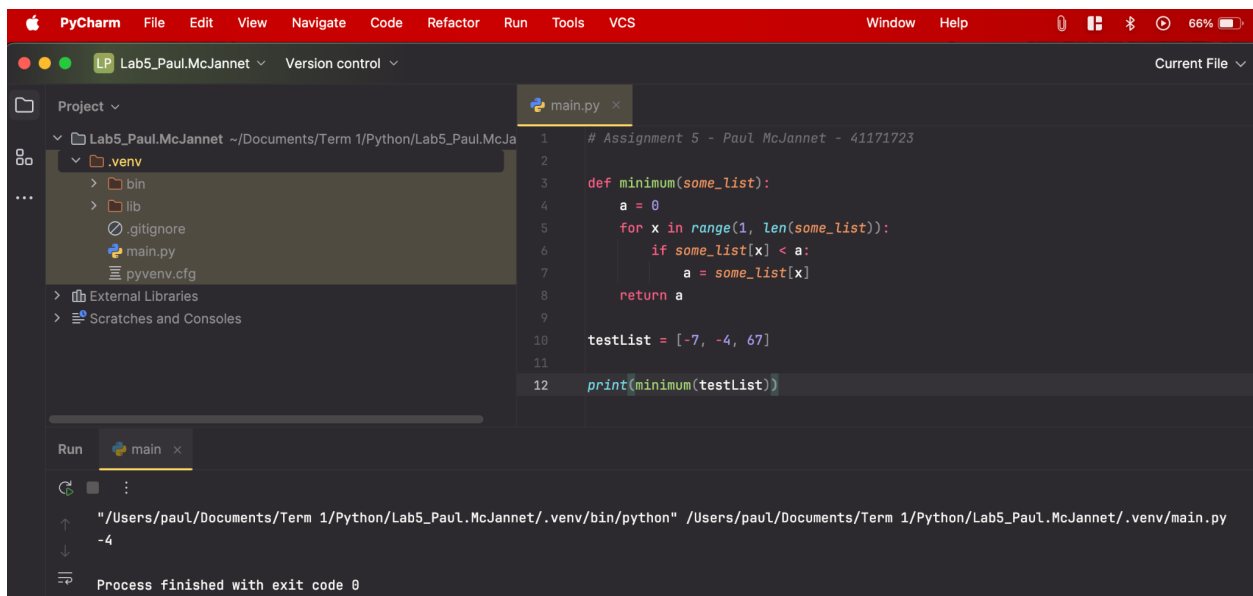


Before we begin to assess our co-workers' code, it might be helpful to understand two Python concepts: what a “list” is and what a “for loop” is. Lists in Python are pretty straightforward: they are a collection that stores data, and are created by using square brackets [] and can separate items by using a comma (W3Schools, 2024a). If you have JavaScript experience, it is similar to how arrays operate.

On the other hand, for loops in Python are used to iterate over some collection of data, such as a list (W3Schools, 2024b). In our co-workers' case, they used the range() function to program when the start of the for loop begins, and when it ends.

Once that is understood, I think it's important to write out the code our co-worker has written and test it with a list of our own (let's call that list “testList” and fill it with some integers):



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The left sidebar shows the project structure for 'Lab5_Paul.McJannet', including a '.venv' directory with 'bin', 'lib', and 'pyvenv.cfg' files, and a 'main.py' file. The main editor window displays the code in 'main.py':

```
1 # Assignment 5 - Paul McJannet - 41171723
2
3 def minimum(some_list):
4     a = 0
5     for x in range(1, len(some_list)):
6         if some_list[x] < a:
7             a = some_list[x]
8     return a
9
10 testList = [-7, -4, 67]
11
12 print(minimum(testList))
```

The bottom panel shows the Run output, indicating that the process finished with exit code 0. The output text is: `"/Users/paul/Documents/Term 1/Python/Lab5_Paul.McJannet/.venv/bin/python" /Users/paul/Documents/Term 1/Python/Lab5_Paul.McJannet/.venv/main.py` followed by `-4`.

After running the code, we can clearly see that something is wrong: in our list -7 is the lowest value and that's what should be returned. However, it is not and -4 is being returned instead.

In order to determine why this is happening, we can step through the loop one iteration at a time and check both what the current iteration is, and what the value of the current index is. In other words, we should check for the value of both “x” and “a” each iteration.

Lets import pdb and use the `set_trace()` function, which will help us see what is going on in more detail. We can place this function call within the loop after our “a” value gets assigned under our conditional. There are two reasons for this. First, we want to put this here because if the condition is not met, the current value in the list is greater than “a” which we don’t care about (as we are trying to find the lowest value). Second, we want to see what the current value of “a” is after it has been assigned, so it should go after the expression “a = some_list[x]”.

By implementing these changes to our co-worker’s code and running the updated program, we can see the following results:

```
Project ▾
  ▾ Lab5_Paul.McJannet ~/Documents/Term 1/Python/Lab5_Pau
    ▾ .venv
      > bin
      > lib
      .gitignore
      main.py
      pyvenv.cfg
    > External Libraries
    > Scratches and Consoles

main.py x
3
4 def minimum(some_list):
5     a = 0
6     for x in range(1, len(some_list)):
7         if some_list[x] < a:
8             a = some_list[x]
9             pdb.set_trace()
10    return a
11
12 testList = [-7, -4, 67]
13
14 print(minimum(testList))

Run main x
"/Users/paul/Documents/Term 1/Python/Lab5_Paul.McJannet/.venv/bin/python" /Users/paul/Documents/Term 1/Python/Lab5_Paul.McJannet/.venv/m
> /Users/paul/Documents/Term 1/Python/Lab5_Paul.McJannet/.venv/main.py(6)minimum()
-> for x in range(1, len(some_list)):
(Pdb) p x
1
(Pdb) p a
-4
(Pdb)
```

After importing and implementing pdb, we are now capable of stepping through the loop using the debugger. A useful command to give pdb is to use the key “p” followed by a variable name. “When using the print command p, you’re passing an expression to be evaluated by Python. If you pass a variable name, pdb prints its current value” (Jennings, n.d., para. 22).

In our case, we first want to evaluate the variable x to see our current loop iteration, so we can command “p x”. We can see that its value is 1. This seems to immediately raise a red flag: since lists are indexed starting at [0], we should be seeing that as our result in the debugger. The fact we are getting 1 means that we are starting that index [1], which is technically the second item in the list! We are completely skipping over the first item and not checking its value!

If we continue to evaluate the value of the current index we can command “p a” and see that the value is -4. This makes sense, because that is indeed the value of the item at index [1] in our testList.

The solution to fixing our co-worker’s problem is to make sure we are including index[0] of our testList. If we observe the code closely, we can see that this issue is related to how the for loop is operating or more specifically, *when* it is starting. The range function is the culprit, because it is taking in two arguments which are telling the loop where to start and where to end. Essentially, by implementing the range() function “the user gets to decide not only where the series of numbers stops but also where it starts, so the user doesn’t have to start at 0 all the time” (GeeksforGeeks, 2024, para. 4). Since our co-worker is starting the range at 1 and not 0, it makes sense that we are starting at index[1] for the loop.

If we correct the start argument in the range() function to be 0, we can run the program and test using the debugger to confirm that our index and its value are what they should be for each loop iteration:

The screenshot shows a code editor with a file named `main.py` open. The code defines a `minimum` function and tests it with a list `testList = [-7, -4, 67]`. The function iterates through the list, updating a variable `a` to the minimum value found. The execution is paused at line 15, `print(minimum(testList))`. The right-hand pane shows the debugger's output, including the current line of code and the state of variables `x` and `a` at each step of the loop.

```
1 # Assignment 5 - Paul McJannet - 41171723
2 import pdb
3
4 def minimum(some_list):
5     a = 0
6     # change 1 to 0 as the first argument in
7     # the range() function
8     for x in range(0, len(some_list)):
9         if some_list[x] < a:
10             a = some_list[x]
11             pdb.set_trace()
12     return a
13
14 testList = [-7, -4, 67]
15 print(minimum(testList))
```

Debugger Output:

```
"/Users/paul/Docu
> /Users/paul/Docu
-> for x in range
(Pdb) p x
0
(Pdb) p a
-7
(Pdb) s
> /Users/paul/Docu
-> if some_list[x
(Pdb) s
> /Users/paul/Docu
-> for x in range
(Pdb) p x
1
(Pdb) p a
-7
(Pdb) s
> /Users/paul/Docu
-> if some_list[x
(Pdb) s
> /Users/paul/Docu
-> for x in range
(Pdb) p x
2
(Pdb) p a
-7
(Pdb) s
> /Users/paul/Docu
-> return a
(Pdb)
```

By using “p x” and “p a” to again check the iteration and current value of “a”, we can step through and now correctly see that -7 is indeed the lowest value of integers in the testList.

Reference List

GeeksforGeeks. (2024, July 25). *Python range() function*. GeeksforGeeks. <https://www.geeksforgeeks.org/python-range-function/>

Jennings, N. (n.d.). *Python Debugging With Pdb*. Real Python. <https://realpython.com/python-debugging-pdb/#printing-expressions>

JetBrains. (2024). *PyCharm* (Version 2024.2.1) [Integrated Development Environment]. <https://www.jetbrains.com/pycharm/>

W3Schools. (2024a). *Python Lists*. W3Schools. https://www.w3schools.com/python/python_lists.asp

W3Schools. (2024b). *Python For Loops*. W3Schools. https://www.w3schools.com/python/python_for_loops.asp