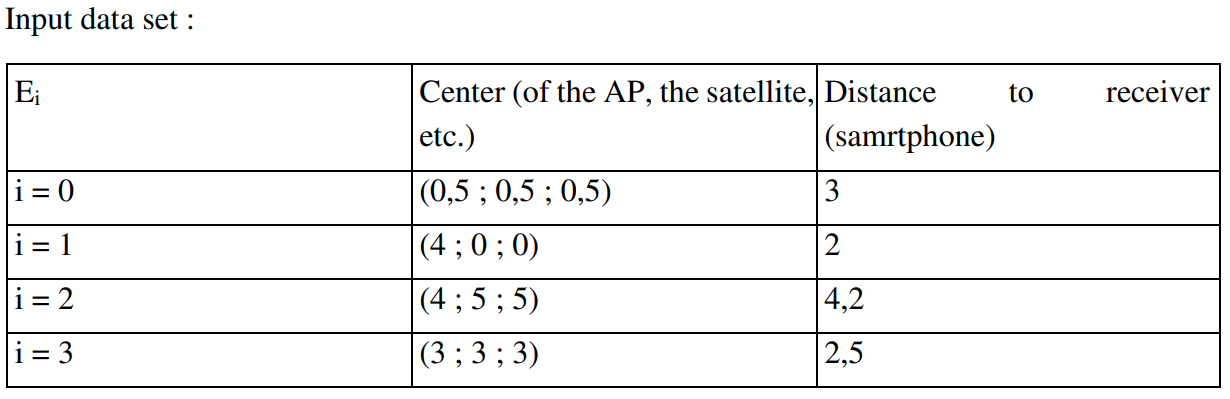
**1: N-Lateration**

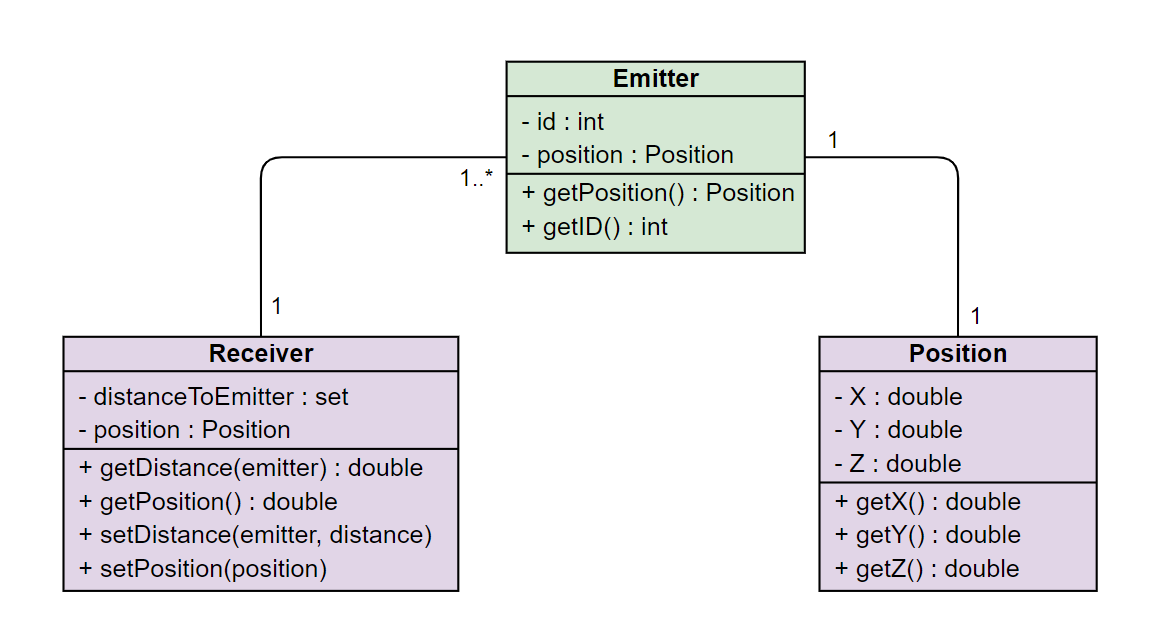
The Language used is Python programming. Taking the values given in the TD Document.



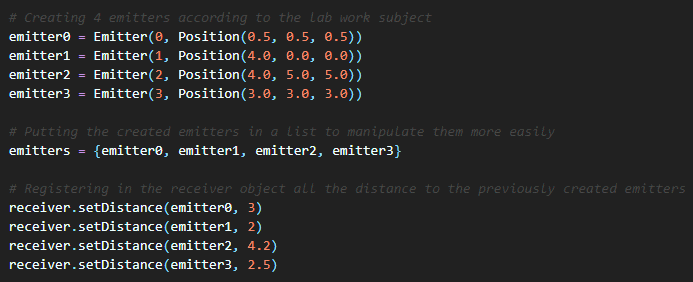
Our script uses 3 classes.

* Position
* Emitter
* Receiver

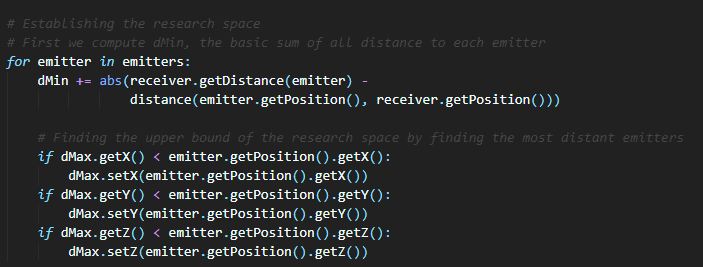
Here you can see the UML class diagram of our code:



We first create 4 emitters with their position and set their distance to the receiver.

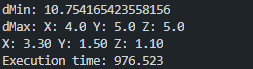


Then we establish the research space and compute dMin which is the sum of all distance to each emitter.

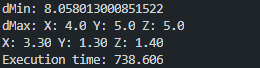


Then we can just find the position which minimize the sum of distance. This position will be the one from our receiver.

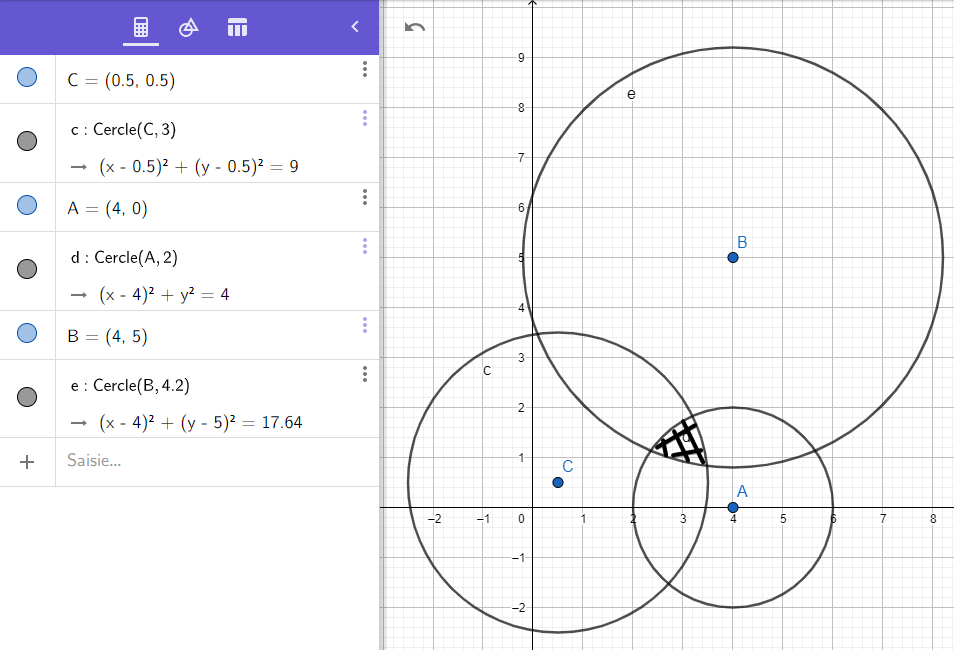
With 4 spheres we get this result:



With only 3 spheres we get this result:



We can verify our results with GeoGebra in 2 dimensions:

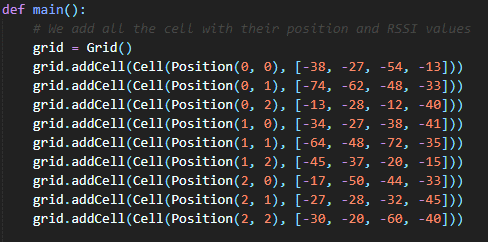


The results look good.

**2: Fingerprint.**

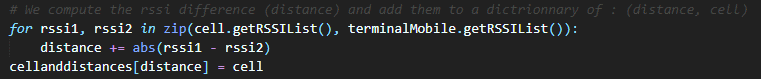
**Goal:**

The goal is to implement a fingerprinting algorithm. We will use 9 cells with their corresponding RSSI values. These values come from the directed work.



We used Python for this lab work as it is a modern and fast programming language that we like. We used 4 classes: Cell, Position, Mobile, Grid. A grid is a composition of Cell and a cell has a Position and a RSSI list. The mobile is the same as a Cell, it has a RSSI list and a position which we will need to find thanks to the fingerprinting algorithm.

To find the K shortest cell we computed the distance between each cell and the mobile terminal. Then we just need to choose the K shortest cell and compute their weight.

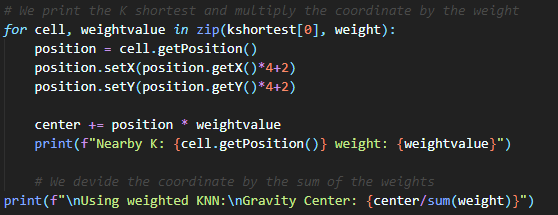


To compute the weight, we used the formula from the lesson:



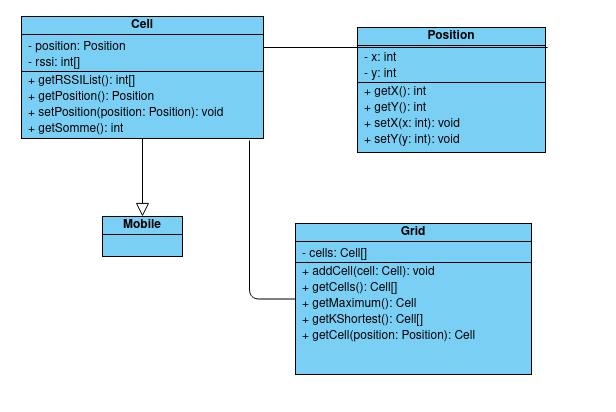


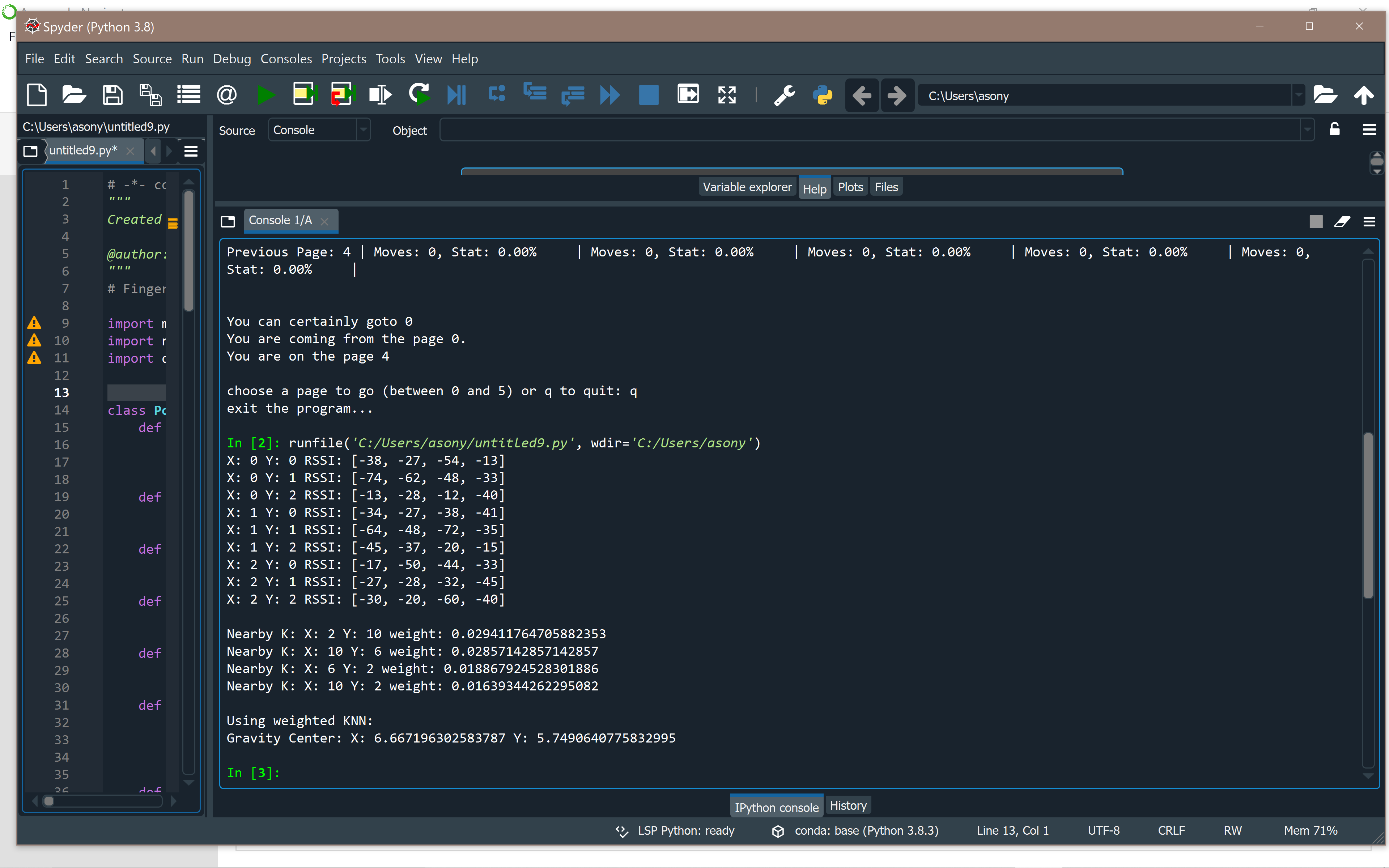
Then we just need to multiply the weight with the position of the cell to calculate the gravity center. Then we can divide the position by the sum of all the weights.



**UML Diagram**

Our script uses different classes that you can see in the diagram below:



 The screen-shot from the output.

**Conclusion**

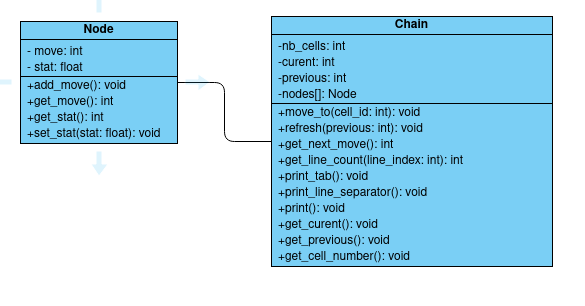
We see that our results are coherent. Here we only have 9 cells so the precision is not very high but if we run this algorithm with a bigger sample of cells and RSSIs we could have an even more precise result.

**3: Hidden Markov Model**

**Goal:**

The goal is to discover what a markov model is, how it works and in what situation it can be used. To understand that we implemented a very simple markov model about the navigation on a website.

**Class Diagram**



A markov model can be interpreted as a graph or a chain of nodes, so we created two classes: Node and Chain. In our case a node represents a web page, and the object contains the number of times the user consulted the page and it’s visit stats. The method add\_move() allows us to increment the number of moves to this node easily.

The chain object contains a list of nodes, the id of the previous visited node, the id of the current node and the total number of nodes in it. There are a lot of methods in this class: the print() method (that use print\_line\_separator() and print\_tab()) is used to print the results of a move on the whole chain as an array in the console. we also have a move\_to() function, that is what its name says, go to a specific node. This method uses the refresh() method to refresh the stat of the previous node after a move. The method get\_next\_move() has the aim to predict the most likely move that will be made by the user.

Results Presentation & Analysis

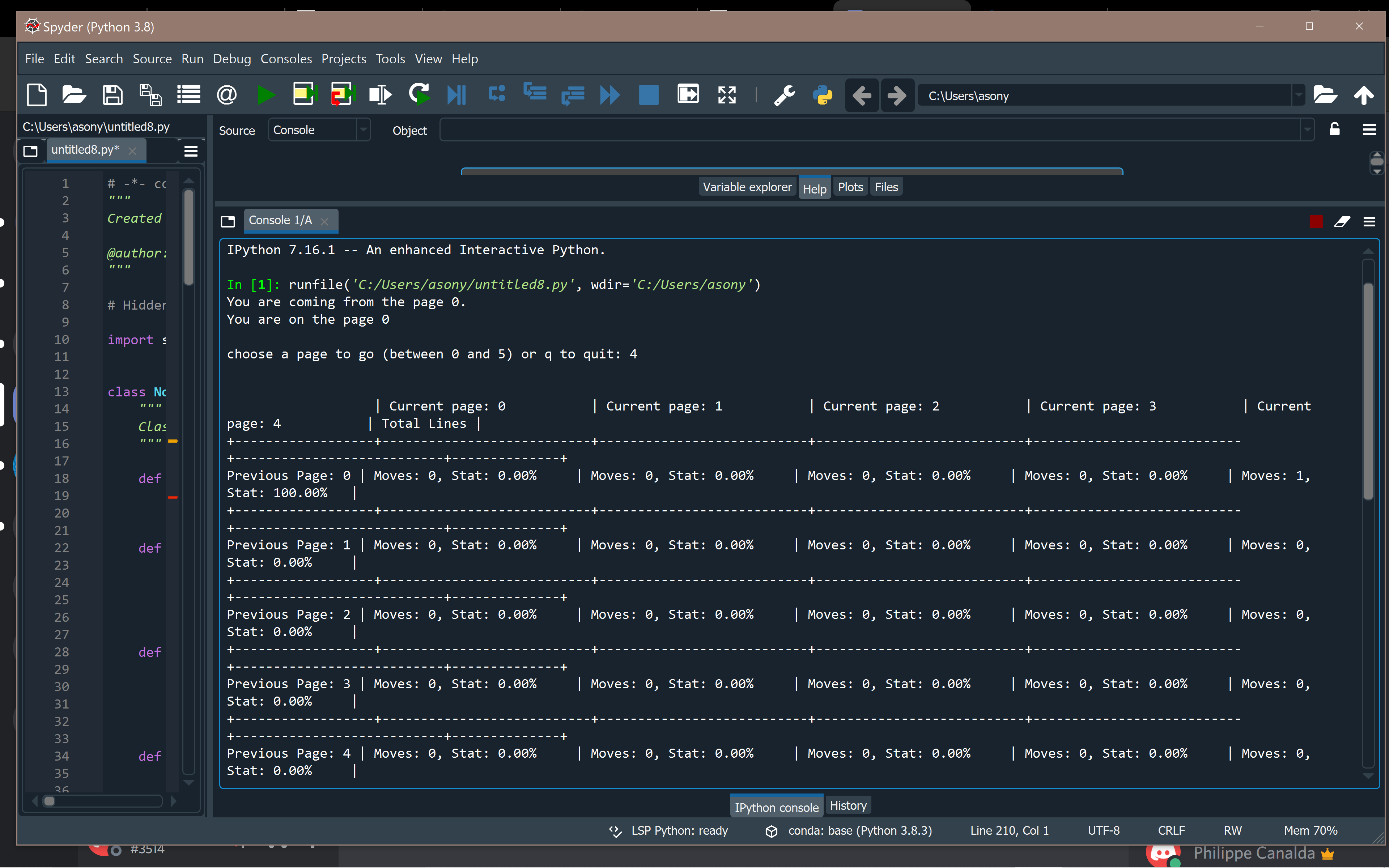
According to the lab work subject, we executed the following sequence:

0 -> 4

0 -> 4 -> 0

0->4->0->0

0->4->0->0->4



We have the expected results.