Implementation of Token Ring Algorithm including Leader Election:

Assumes that processes are logically ordered in some fashion, and that each process knows the order and who is coordinator. No token is involved. When a process notices that the coordinator is not responding it sends an ELECTION message with its own id to its downstream neighbor. If that neighbor doesn't respond it sends it to its neighbor's neighbor, etc. Each station that receives the ELECTION message adds its own id to the list. When the message circulates back to the originator it selects the highest id in the list and sends a COORDINATOR message announcing the new coordinator. This message circulates once and is removed by the originator.
If two elections are held simultaneously (say because two different processes notice simultaneously that the coordinator is dead) then each comes up with the same list and elects the same coordinator. Some time is wasted, but nothing is really hurt by this.

This code also contains the logic for electing a new leader if the orignal cordinator goes down.

How to run this code

Single election:

Run **node4.py, node1.py, node2.py, node3.py** in this order. Look at the output's of node 4. You will see token been passed at high speed. Than close the node 4 program. You will see leader election take place. You will see node 3 become the leader. Than again start node 4, you will see node 4 start election and leader election take place.

Multiple election:

Please go to node 2 and comment the timeout from 21. Uncomment command for timeout for 15. Without this multiple election won't work. Run **node4.py, node1.py, node2.py, node3.py** in this order. Look at the output's of node 4. You will see token been passed at high speed. Than close the node 4 program. You will see leader election take place by both node 1 and node 2. You will see node 3 become the leader. Than again start node 4, you will see node 4 start election and leader election take place.