|  |  |
| --- | --- |
| class Pro:  def \_\_init\_\_(self,id):  self.id = id  self.act = True  class RING:  def \_\_init\_\_(self):  self.total\_pro = 8  self.pro\_list = []  def initiate\_list(self):  self.pro\_list = [Pro(i) for i in range(0,self.total\_pro)]  def election(self):  print("if",self.pro\_list[self.fetchMax()].id,"fails")  self.pro\_list[self.fetchMax()].act = False  intiator = 2  print("Election started by 2")  old = intiator  new = (old+1)%8  while(new!=intiator):  if(self.pro\_list[new].act == True):  print("process"+str(self.pro\_list[old].id)+"pass election to "+str(self.pro\_list[new].id))  old = new  new = (new+1)%8  coord = str(self.fetchMax())  print("the new coordinator after the election is : "+coord)  curr = (int(coord) +1)%8  while(curr!=int(coord)):  print(str(self.pro\_list[curr].id)+" got the election result of "+coord+" being the new coordinator")  curr = (curr+1)%8  print("End of the election")  def fetchMax(self):  max = -999  i = 0  ele =0  for i in range(0,8):  if(self.pro\_list[i].id>max and self.pro\_list[i].act):  max = self.pro\_list[i].id  return max  def main():  obj = RING()  obj.initiate\_list()  obj.election()  if \_\_name\_\_ == "\_\_main\_\_":  main() | class Node:  def \_\_init\_\_(self, node\_id):  self.node\_id = node\_id  self.leader = None  def \_\_str\_\_(self):  return f"Node {self.node\_id}"  def start\_election(self, nodes):  print(f"{self} initiates an election.")  higher\_nodes = [node for node in nodes if node.node\_id > self.node\_id]  for higher\_node in higher\_nodes:  higher\_node.receive\_election(self)  def receive\_election(self, initiating\_node):  print(f"{self} receives an election message from {initiating\_node}.")  if self.leader is None or self.node\_id > self.leader.node\_id:  print(f"{self} sends an OK message to {initiating\_node}.")  initiating\_node.receive\_ok(self)  else:  print(f"{self} ignores the election message from {initiating\_node}.")  def receive\_ok(self, responding\_node):  print(f"{self} receives an OK message from {responding\_node}.")  self.leader = responding\_node  print(f"{self} acknowledges {responding\_node} as the leader.")  if \_\_name\_\_ == "\_\_main\_\_":  nodes = [Node(1), Node(2), Node(3), Node(4), Node(5), Node(6), Node(7)]  for node in nodes:  node.start\_election(nodes)  elected\_leader = max(nodes, key=lambda node: node.node\_id)  print(f"\n{elected\_leader} is elected as the leader.") |