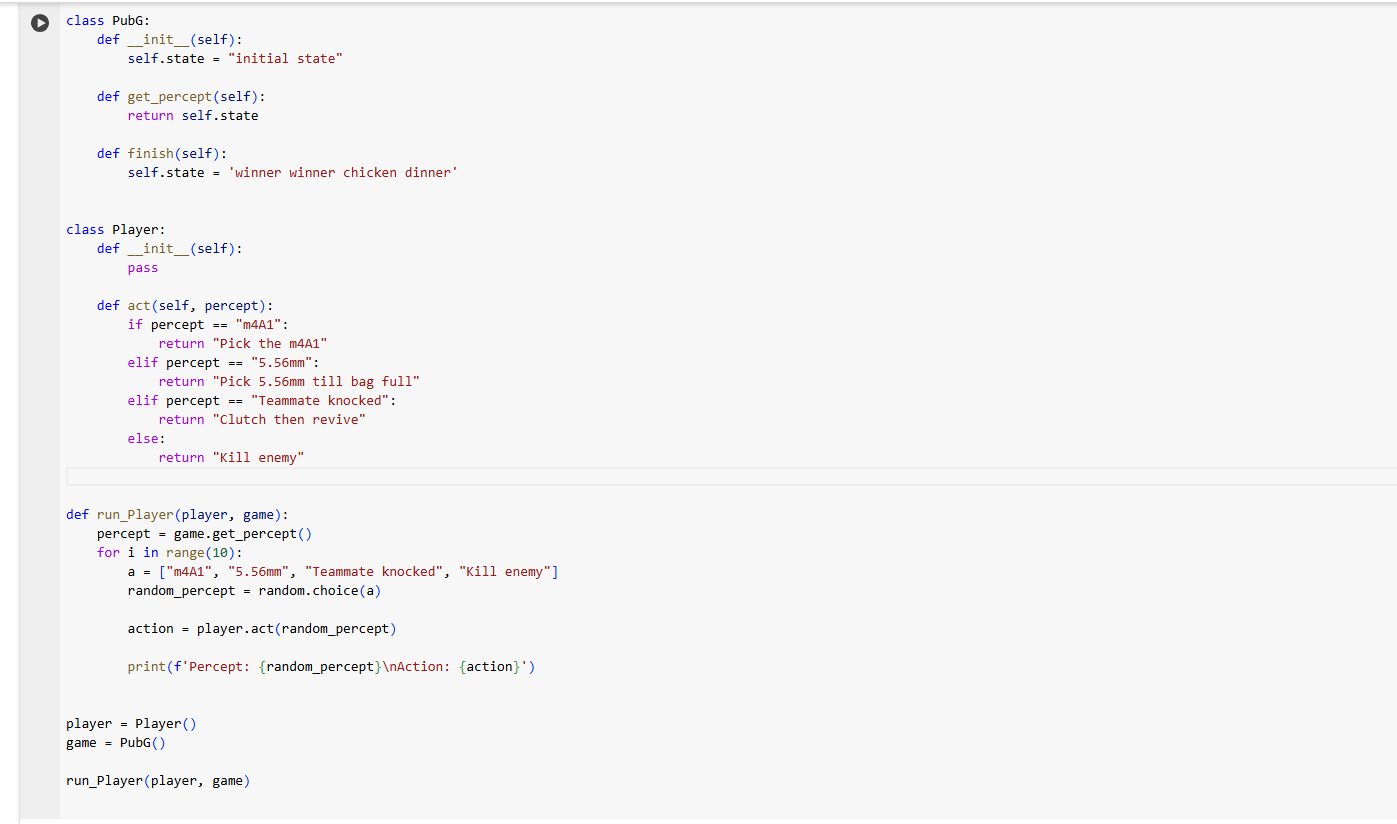
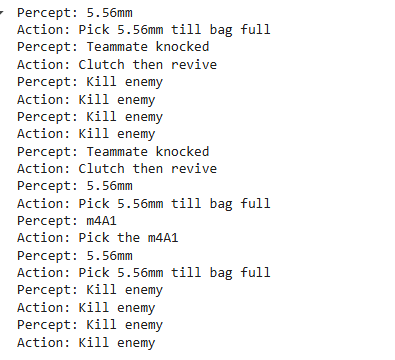
**Class Task**

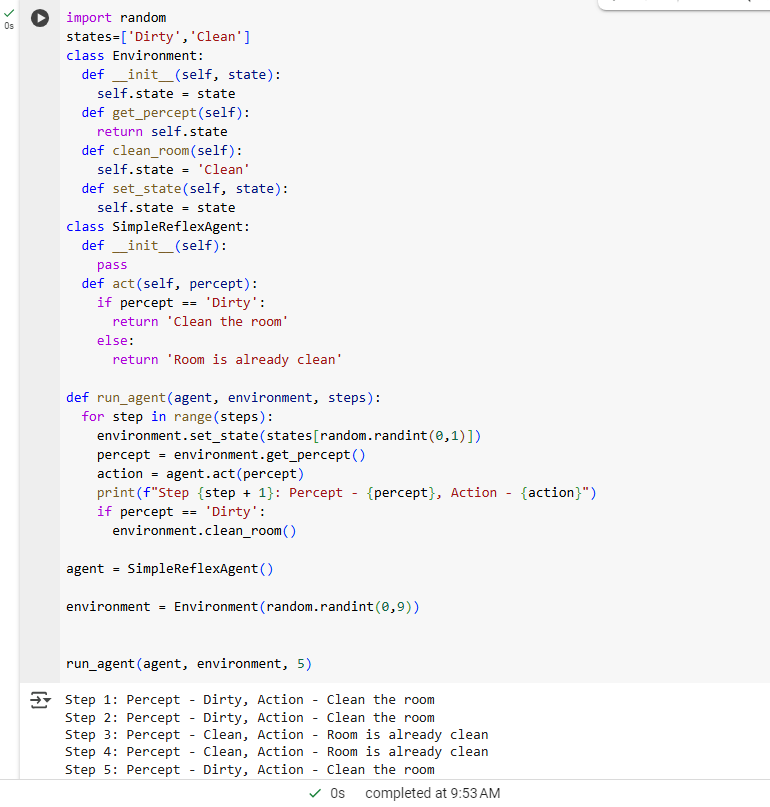




Q1:



Q2:



**Lab Task**

Task 1:

import random

class Enivronment:

def \_\_init\_\_(self,state = 'Initial',component = None):

self.state = state

if component is None:

component = []

for i in range(9):

component.append(random.choice(['vulnerable','safe']))

self.component = component

def display(self):

print(f'\nState : {self.state}\n\n')

var = 'A'

for i in range (9):

print(f"Component {var}: {self.component[i]}")

var = chr(ord(var) + 1)

class Agent:

def \_\_init\_\_(self):

pass

def scan(self,env):

print('\n\nScanning components\n\n')

var = 'A'

for i in range(9):

if env.component[i] == 'vulnerable':

print(f'Component {var} -> Warning : Vulnerability Found')

else:

print(f'Component {var} -> Scan completed with success')

var = chr(ord(var)+1)

def patch(self,env):

print('\n\nPatching components\n\n')

var = 'A'

for i in range(9):

if env.component[i] == 'vulnerable':

print(f'Component {var} -> PATCHING')

env.component[i] = 'safe'

var = chr(ord(var)+1)

env.state = 'Final'

env = Enivronment()

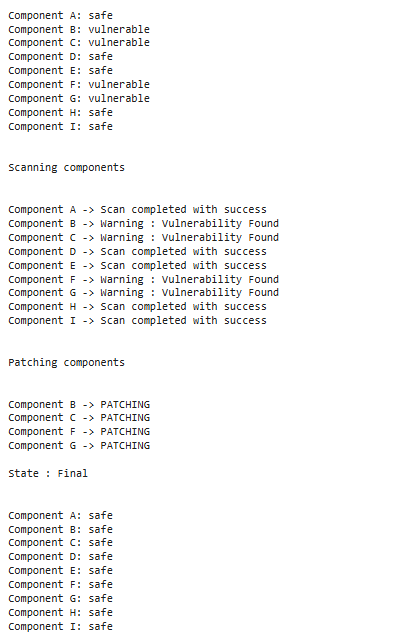
env.display()

agent = Agent()

agent.scan(env)

agent.patch(env)

env.display()



Task 2:

import random

class Server:

def \_\_init\_\_(self, id):

self.id = id

self.load = random.randint(1, 10)

def get\_status(self):

if self.load < 5:

return "Underloaded"

elif self.load < 8:

return "Balanced"

else:

return "Overloaded"

class Agent:

def \_\_init\_\_(self, servers):

self.servers = servers

def balance\_load(self):

i = 0

while i < len(self.servers):

if self.servers[i].get\_status() == "Overloaded":

j = 0

while j < len(self.servers):

if i != j and self.servers[j].get\_status() == "Underloaded":

self.servers[i].load -= 1

self.servers[j].load += 1

else:

j += 1

i += 1

def display\_status(self):

for server in self.servers:

print(f"Server {server.id}: Load {server.load}, Status: {server.get\_status()}")

servers = [Server(i) for i in range(1, 6)]

agent = Agent(servers)

print("Initial Server Load Status:")

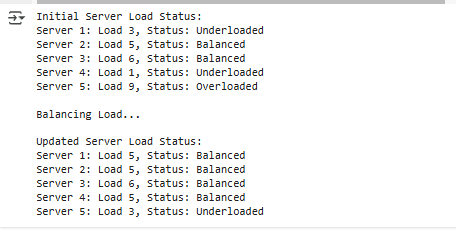
agent.display\_status()

print("\nBalancing Load...")

agent.balance\_load()

print("\nUpdated Server Load Status:")

agent.display\_status()



Task 3:

import random

class BackupSystem:

def \_\_init\_\_(self, state='Initial', tasks=None):

self.state = state

if tasks is None:

tasks = []

for i in range(9):

tasks.append(random.choice(['Completed', 'Failed']))

self.tasks = tasks

def display(self):

print(f'\nState: {self.state}\n')

for i in range(9):

print(f"Task {i+1}: {self.tasks[i]}")

class BackupAgent:

def \_\_init\_\_(self):

pass

def scan(self, system):

print('\n\nScanning backup tasks\n')

for i in range(9):

if system.tasks[i] == 'Failed':

print(f'Task {i+1} -> Warning: Backup Failed')

else:

print(f'Task {i+1} -> Backup Successful')

def retry(self, system):

print('\n\nRetrying failed backups\n')

for i in range(9):

if system.tasks[i] == 'Failed':

print(f'Task {i+1} -> RETRYING')

system.tasks[i] = 'Completed'

system.state = 'Final'

backup\_system = BackupSystem()

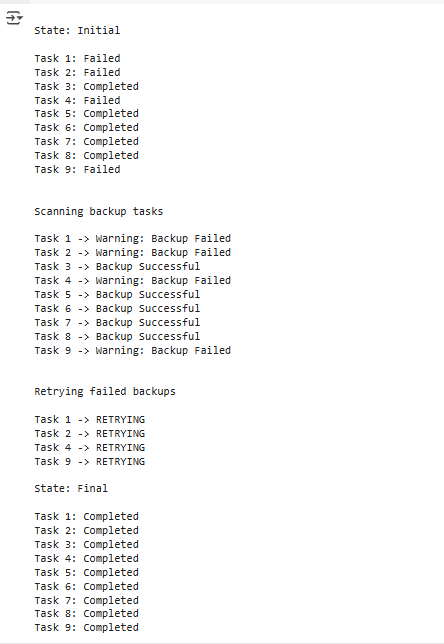
backup\_system.display()

backup\_agent = BackupAgent()

backup\_agent.scan(backup\_system)

backup\_agent.retry(backup\_system)

backup\_system.display()



Task 4:

import random

class Enivronment:

def \_\_init\_\_(self,state = 'Initial',component = None):

self.state = state

if component is None:

component = []

for i in range(9):

component.append(random.choice(['low risk vulnerable','high risk vulnerable','safe']))

self.component = component

def display(self):

print(f'\nState : {self.state}\n\n')

var = 'A'

for i in range (9):

print(f"Component {var}: {self.component[i]}")

var = chr(ord(var) + 1)

class Agent:

def \_\_init\_\_(self):

pass

def scan(self,env):

print('\n\nScanning components\n\n')

var = 'A'

for i in range(9):

if env.component[i] == 'low risk vulnerable' or env.component[i] == 'high risk vulnerable':

print(f'Component {var} -> Warning : Vulnerability Found')

else:

print(f'Component {var} -> Scan completed with success')

var = chr(ord(var)+1)

def patch(self,env):

print('\n\nPatching components\n\n')

var = 'A'

for i in range(9):

if env.component[i] == 'low risk vulnerable':

print(f'Component {var} -> PATCHING')

env.component[i] = 'safe'

var = chr(ord(var)+1)

env.state = 'Final'

env = Enivronment()

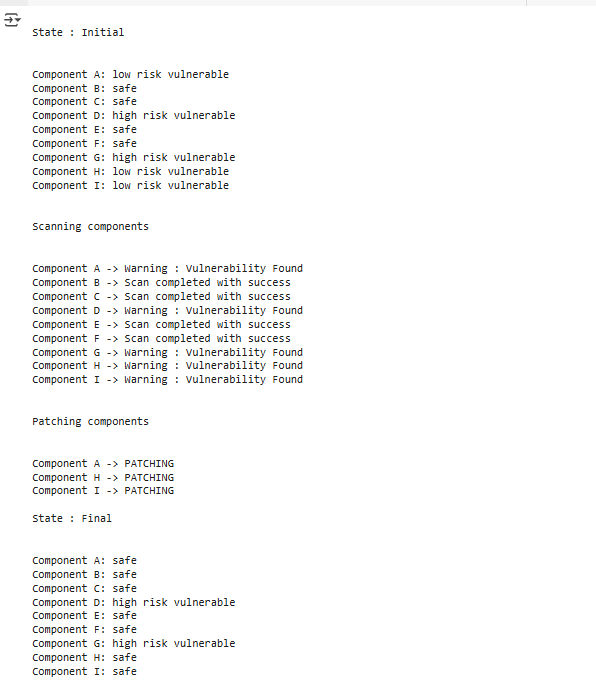
env.display()

agent = Agent()

agent.scan(env)

agent.patch(env)

env.display()



Task 5:

import random

class Hospital:

def \_\_init\_\_(self):

self.room = [1, 2, 3, 4]

self.patient\_id = [101, 102, 103, 104]

self.medicine = [False, False, True, False]

def get\_status(self, id):

for i in range(len(self.patient\_id)):

if id == self.patient\_id[i]:

return self.medicine[i]

return None

def get\_room(self, id):

for i in range(len(self.patient\_id)):

if id == self.patient\_id[i]:

return self.room[i]

return None

class Agent:

def \_\_init\_\_(self):

pass

def act(self, hos, id, room, idx):

print(f'Robot : Moving to storage area\n')

print(f'Robot : Picking up medicine\n')

print(f'Robot : Ready to deliver medicine\n')

print(f'Robot : Scanning patient id {id}\n')

if id == hos.patient\_id[idx] and room == hos.room[idx]:

print('Robot : Scanning completed successfully\n')

print(f'Robot : Delivering medicine to patient {id}\n')

temp = random.randint(0, 1)

if temp == 0:

print('Robot : Situation is out of hand, alerting staff\n')

else:

print('Robot : Situation is in control\n\n')

def run\_agent(agent, hospital):

for idx, id in enumerate(hospital.patient\_id):

room = hospital.get\_room(id)

medicine\_needed = hospital.get\_status(id)

if not medicine\_needed:

agent.act(hospital, id, room, idx)

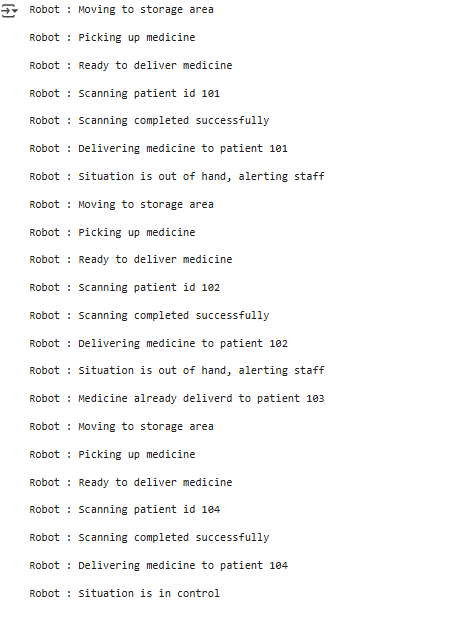
else:

print(f'Robot : Medicine already deliverd to patient {id}\n')

environment = Hospital()

agent = Agent()

run\_agent(agent, environment)



Task 6:

class Building:

def \_\_init\_\_(self):

self.grid = [' ', ' ', '🔥',

' ', '🔥', ' ',

' ', ' ', '🔥']

def sense\_fire(self, position):

return self.grid[position] == '🔥'

def extinguish\_fire(self, position):

if self.grid[position] == '🔥':

self.grid[position] = ' '

return "Fire extinguished! 🚒"

return "No fire here."

def display\_status(self):

print("\nBuilding Status:")

for i in range(0, 9, 3):

print(" | ".join(self.grid[i:i+3]))

print()

class FireFighterRobot:

def \_\_init\_\_(self):

self.position = 0

self.path = [0, 1, 2, 3, 4, 5, 6, 7, 8]

def move(self):

if self.position < len(self.path) - 1:

self.position += 1

def operate(self, building):

for step in range(len(self.path)):

print(f"\nStep {step + 1}: Robot at Position {self.position + 1}")

if building.sense\_fire(self.position):

print("🔥 Fire detected! Extinguishing...")

print(building.extinguish\_fire(self.position))

else:

print("✅ No fire detected. Moving ahead.")

building.display\_status()

self.move()

building = Building()

robot = FireFighterRobot()

robot.operate(building)



