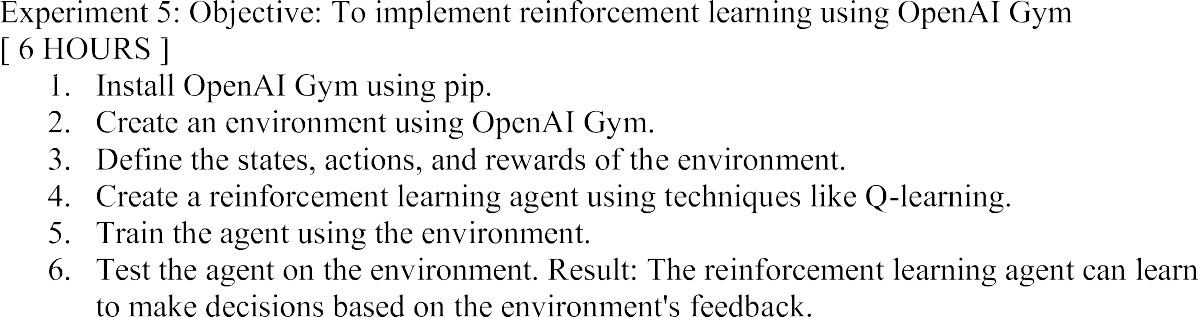
#### Lab Assignment 9

#### Name: M Jeyapathy

#### usn: 22btrad016

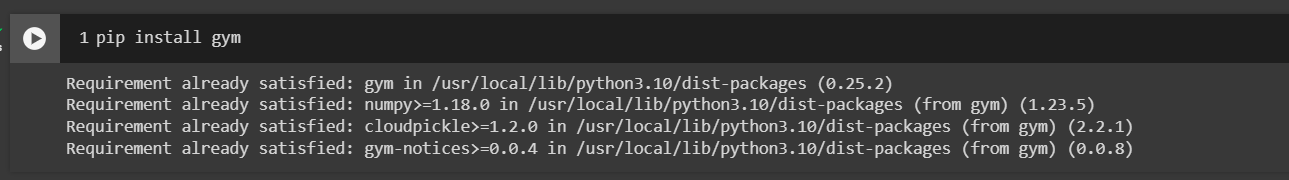
#### Question 1:



1. Install OpenAI Gym using pip.

CODE:

pip install gym



1. Create an environment using OpenAI Gym.

CODE:

import gym

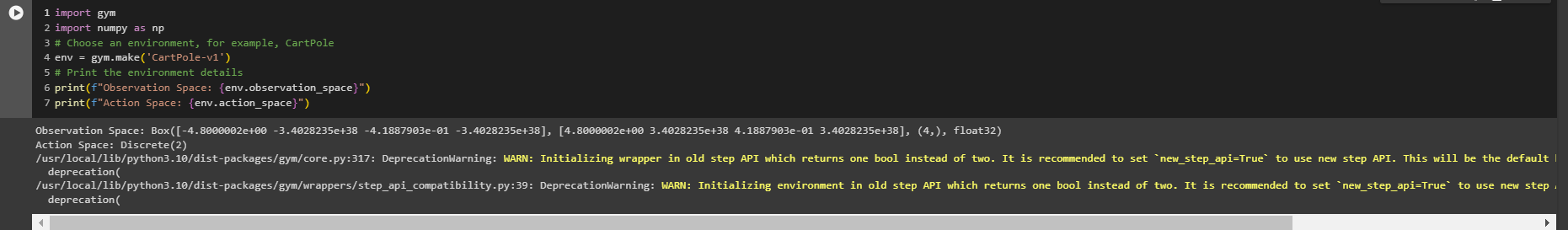
import numpy as np

# Choose an environment, for example, CartPole

env = gym.make('CartPole-v1')

# Print the environment details

print(f"Observation Space: {env.observation\_space}")

print(f"Action Space: {env.action\_space}")

1. Define the states, actions, and rewards of the environment

**CODE:**

# Inspect the state and action spaces

print(f"Observation Space Low: {env.observation\_space.low}")

print(f"Observation Space High: {env.observation\_space.high}")

print(f"Action Space N: {env.action\_space.n}")

# Run a few episodes to understand the reward structure

for \_ in range(3):

state = env.reset()

total\_reward = 0

while True:

# Random action for exploration

action = env.action\_space.sample()

next\_state, reward, done, \_ = env.step(action)

total\_reward += reward

# Print the state, action, reward for each step

print(f"State: {state}, Action: {action}, Reward: {reward}, Done: {done}")

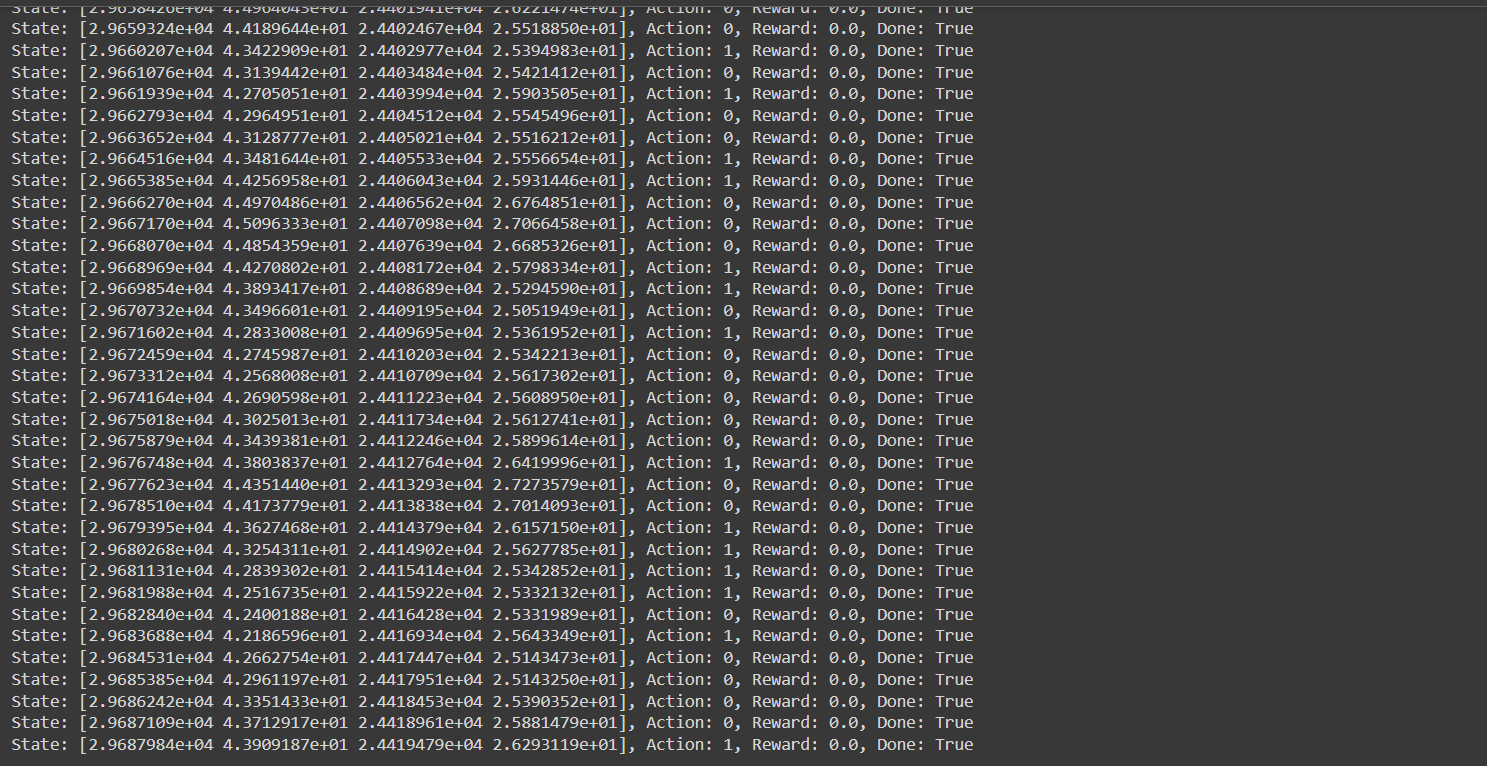
state = next\_state

if done:

break

print(f"Total Reward: {total\_reward}")





1. Create a reinforcement learning agent using techniques like Q-learning.

**CODE:**

import numpy as np

class QLearningAgent:

def \_\_init\_\_(self, state\_space\_size, action\_space\_size, learning\_rate=0.1, discount\_factor=0.99,

exploration\_prob=1.0, exploration\_decay=0.995):

self.learning\_rate = learning\_rate

self.discount\_factor = discount\_factor

self.exploration\_prob = exploration\_prob

self.exploration\_decay = exploration\_decay

self.action\_space\_size = action\_space\_size

# Discretize the state space

self.state\_space\_bins = [np.linspace(env.observation\_space.low[i], env.observation\_space.high[i],

num=10) for i in range(state\_space\_size)]

# Initialize Q-table with discrete state space

self.q\_table = np.zeros(tuple(len(bins) - 1 for bins in self.state\_space\_bins) + (action\_space\_size,))

def discretize\_state(self, state):

return tuple(np.digitize(state[i], self.state\_space\_bins[i]) - 1 for i in range(len(state)))

def choose\_action(self, state):

state = self.discretize\_state(state)

if np.random.rand() < self.exploration\_prob:

return np.random.choice(self.action\_space\_size)

else:

return np.argmax(self.q\_table[state])

def update\_q\_table(self, state, action, reward, next\_state):

state = self.discretize\_state(state)

next\_state = self.discretize\_state(next\_state)

best\_next\_action = np.argmax(self.q\_table[next\_state])

self.q\_table[state + (action,)] += self.learning\_rate \* (reward + self.discount\_factor \*

self.q\_table[next\_state + (best\_next\_action,)] - self.q\_table[state + (action,)])

# Decay exploration probability

self.exploration\_prob \*= self.exploration\_decay

# Assuming you know the state and action space sizes

state\_space\_size = env.observation\_space.shape[0]

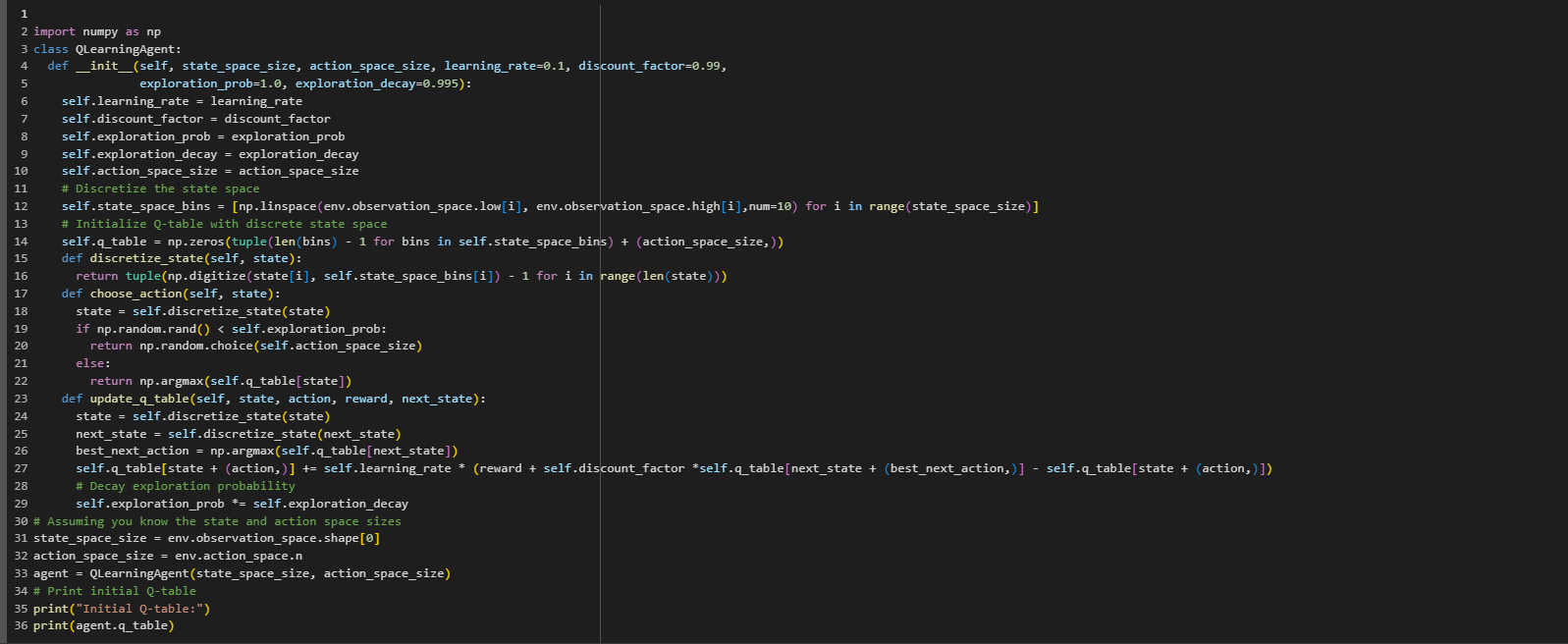
action\_space\_size = env.action\_space.n

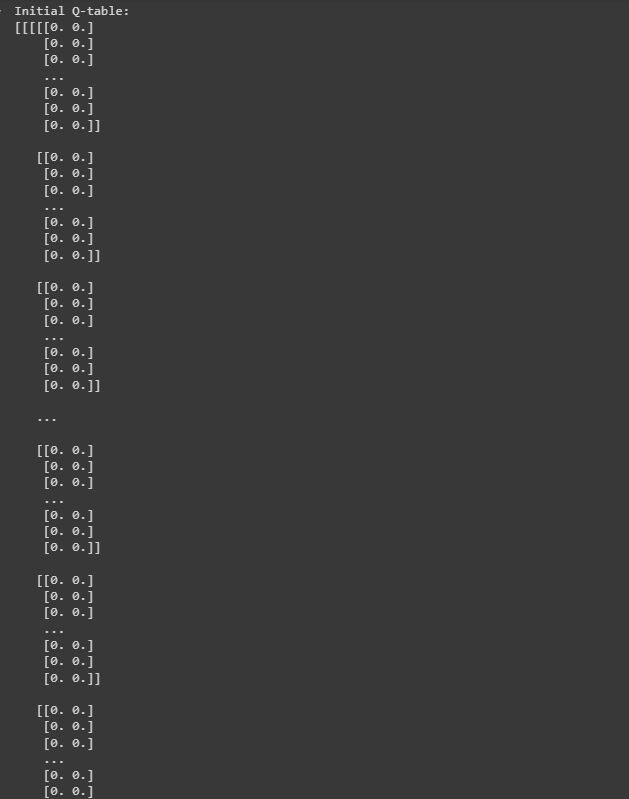
agent = QLearningAgent(state\_space\_size, action\_space\_size)

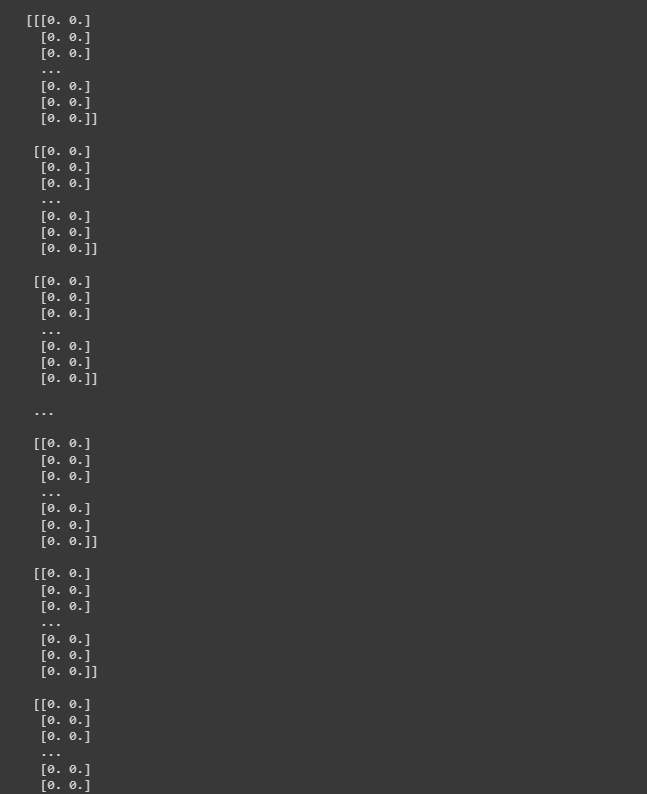
# Print initial Q-table

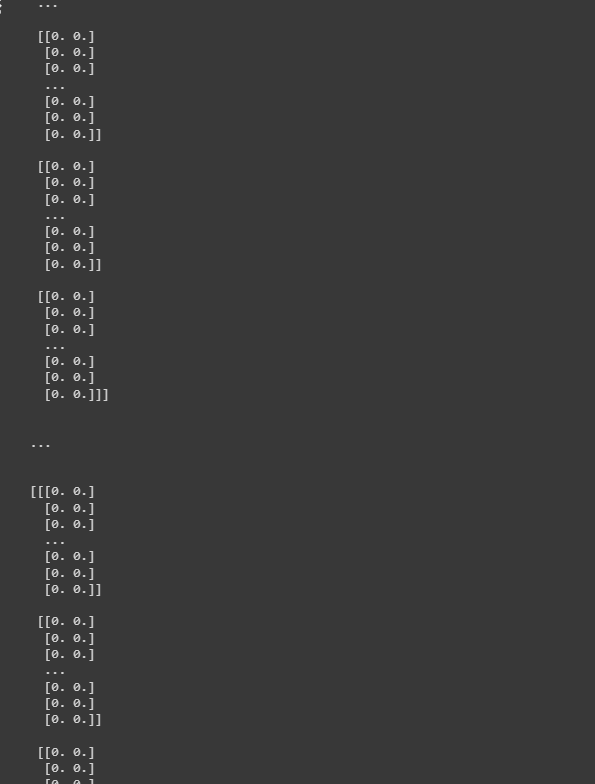
print("Initial Q-table:")

print(agent.q\_table)









1. Train the agent using the environment**.**

**CODE:**

num\_episodes = 1000

for episode in range(num\_episodes):

state = env.reset()

total\_reward = 0

while True:

action = agent.choose\_action(state)

next\_state, reward, done, \_ = env.step(action)

agent.update\_q\_table(state, action, reward, next\_state)

total\_reward += reward

state = next\_state

if done:

break

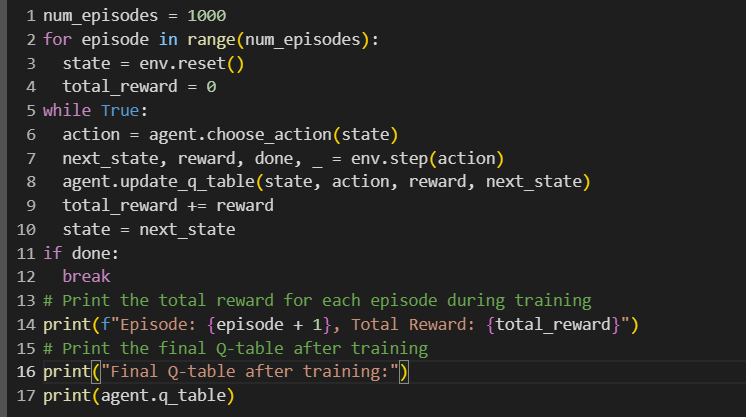
# Print the total reward for each episode during training

print(f"Episode: {episode + 1}, Total Reward: {total\_reward}")

# Print the final Q-table after training

print("Final Q-table after training:")

print(agent.q\_table)









6. Test the agent on the environment. Result: The reinforcement learning agent can learn to make decisions based on the environment's feedback.

**CODE:**

num\_test\_episodes = 10

for episode in range(num\_test\_episodes):

state = env.reset()

total\_reward = 0

while True:

action = agent.choose\_action(state)

next\_state, reward, done, \_ = env.step(action)

total\_reward += reward

state = next\_state

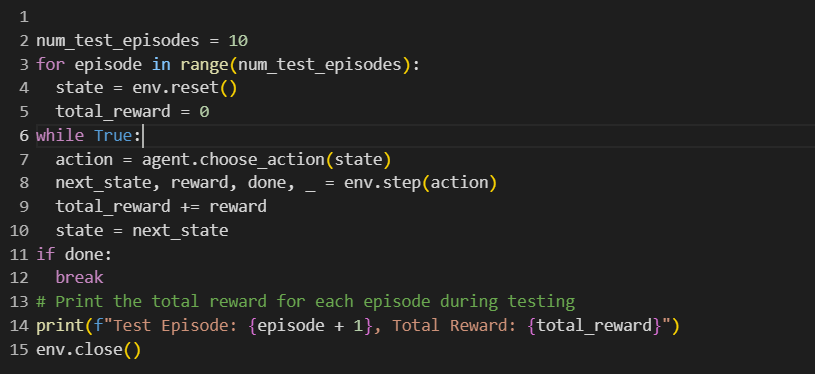
if done:

break

# Print the total reward for each episode during testing

print(f"Test Episode: {episode + 1}, Total Reward: {total\_reward}")

env.close()



Github Link:

https://github.com/Jeyapathy/Machine-Learning