Faculty of Technology and Engineering

Chandubhai S. Patel Institute of Technology

Date: / /

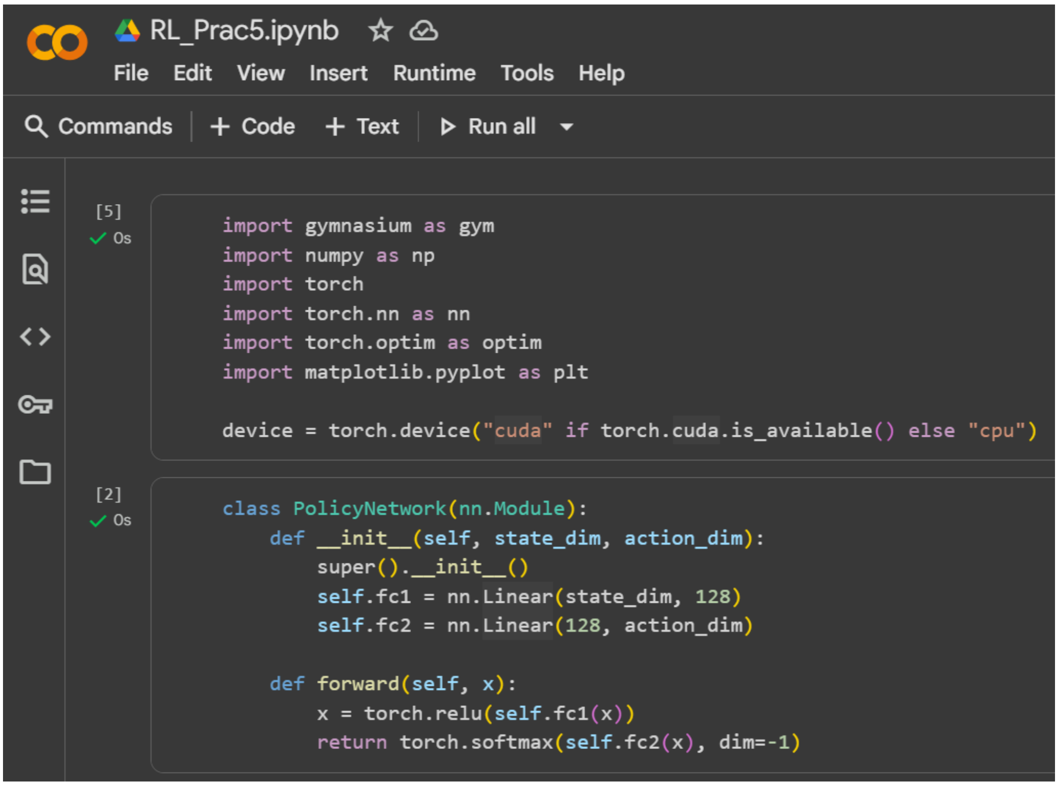
Practical Performa

Academic Year	:	2025-26	Semester	:	7th
Course code	:	OCCSE4001	Course name	:	Reinforcement Learning

Practical- No. 5

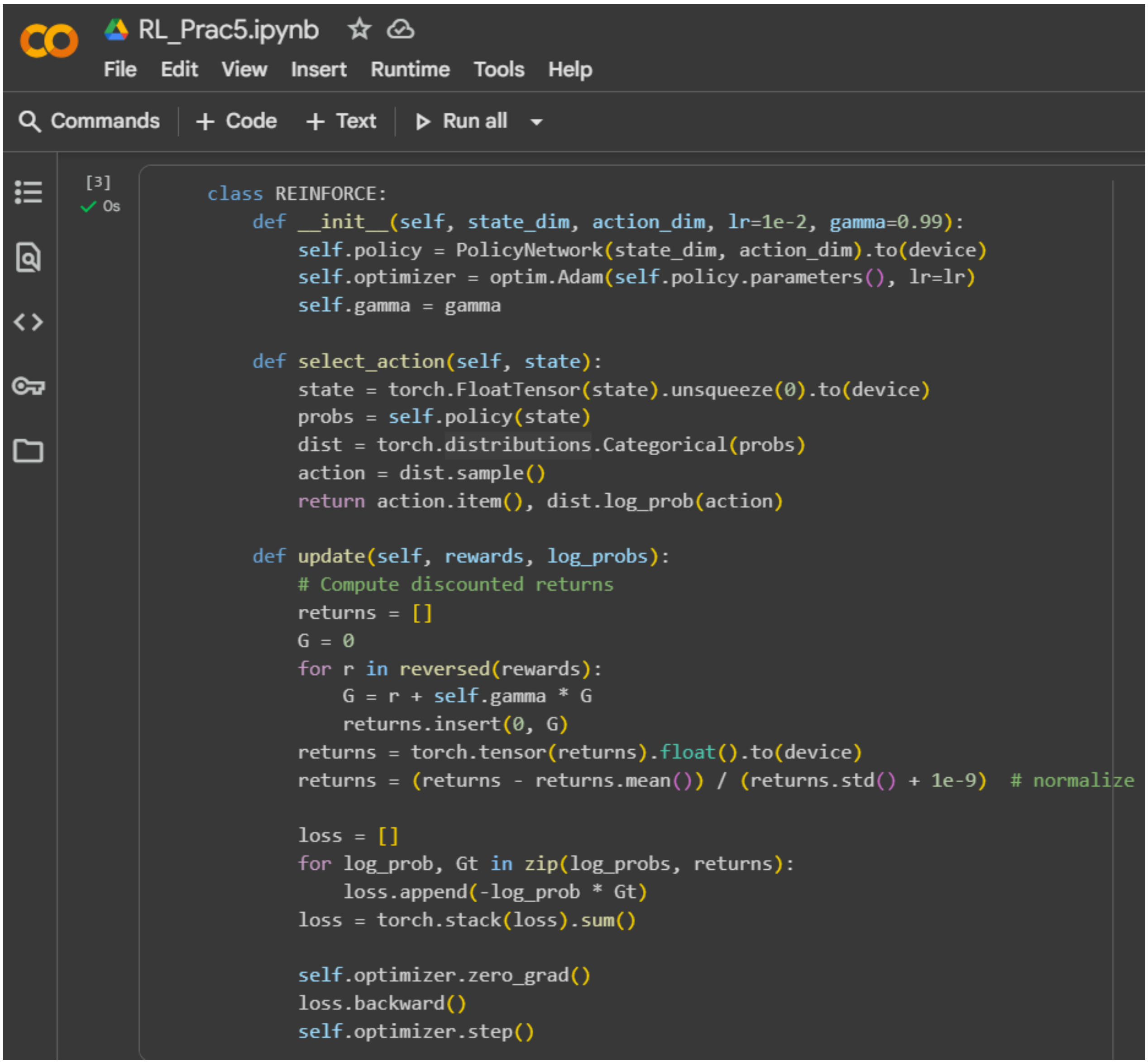
Aim: To implement the REINFORCE algorithm and optimize a stochastic policy using gradient ascent.

Code:

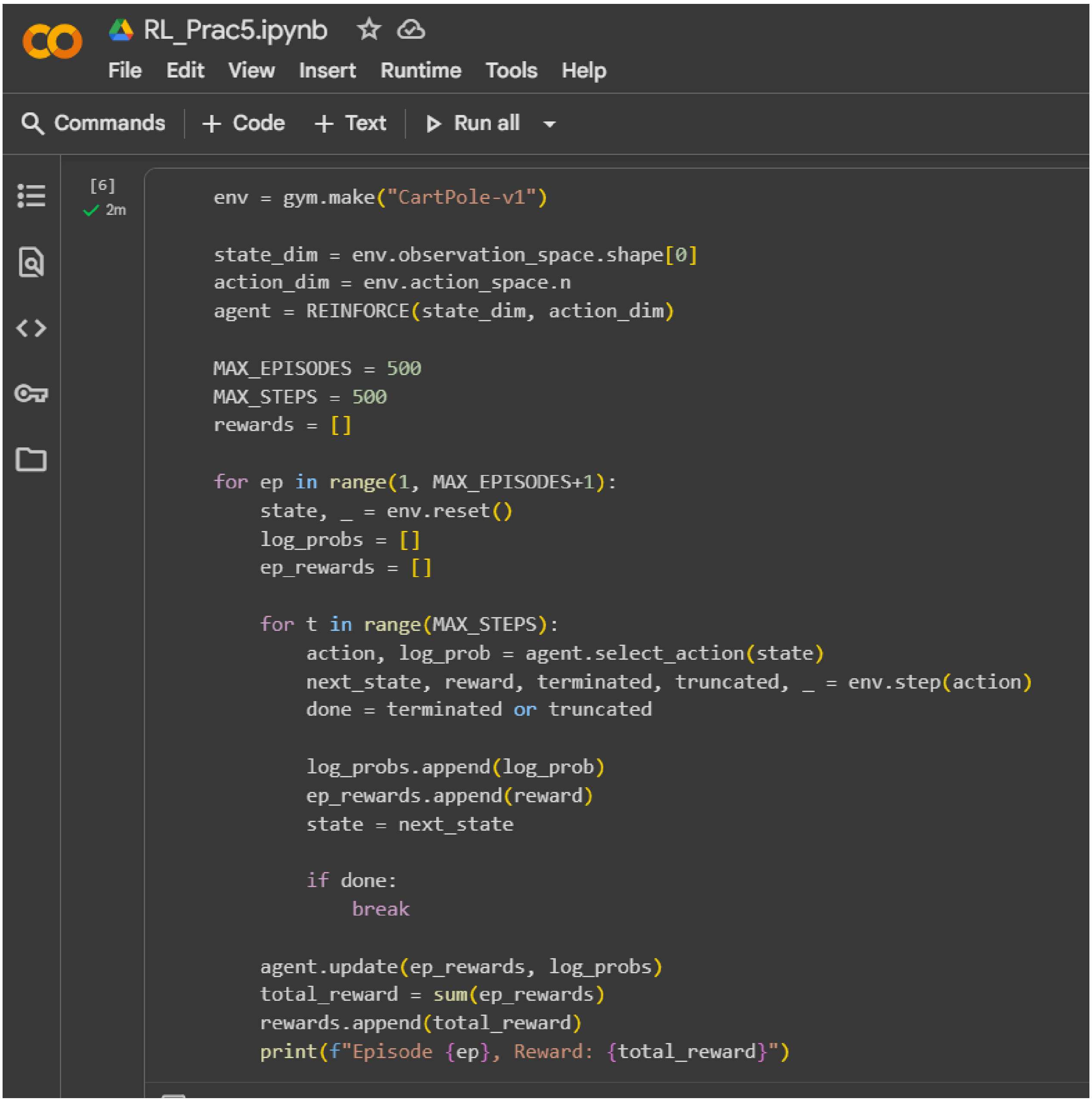


```python
import gymnasium as gym
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

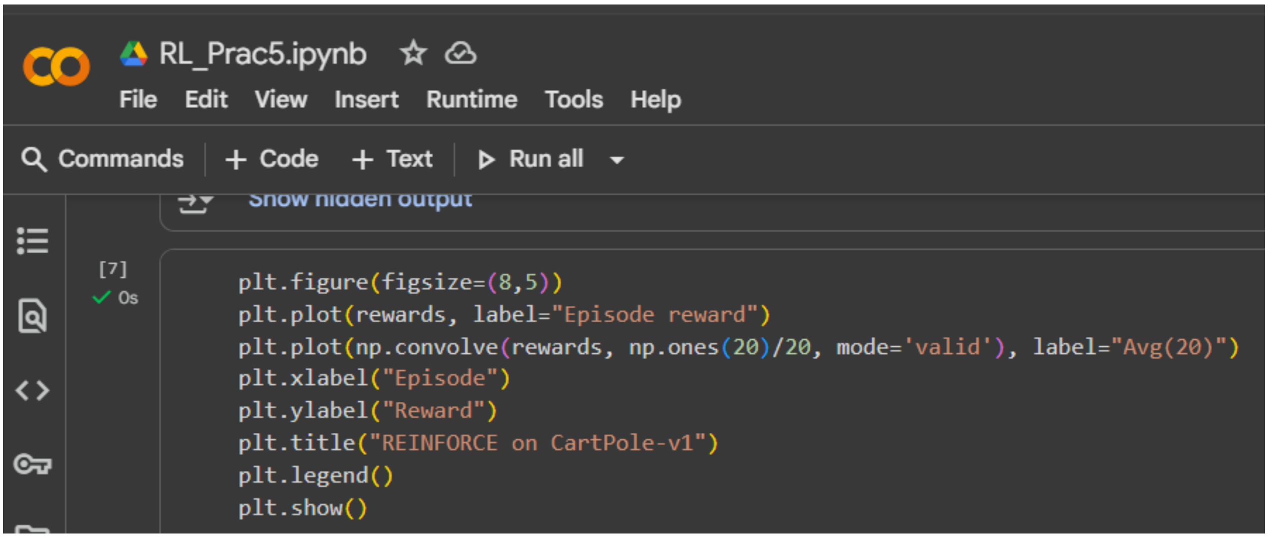
```python
class PolicyNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.fc1 = nn.Linear(state_dim, 128)
        self.fc2 = nn.Linear(128, action_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return torch.softmax(self.fc2(x), dim=-1)
```



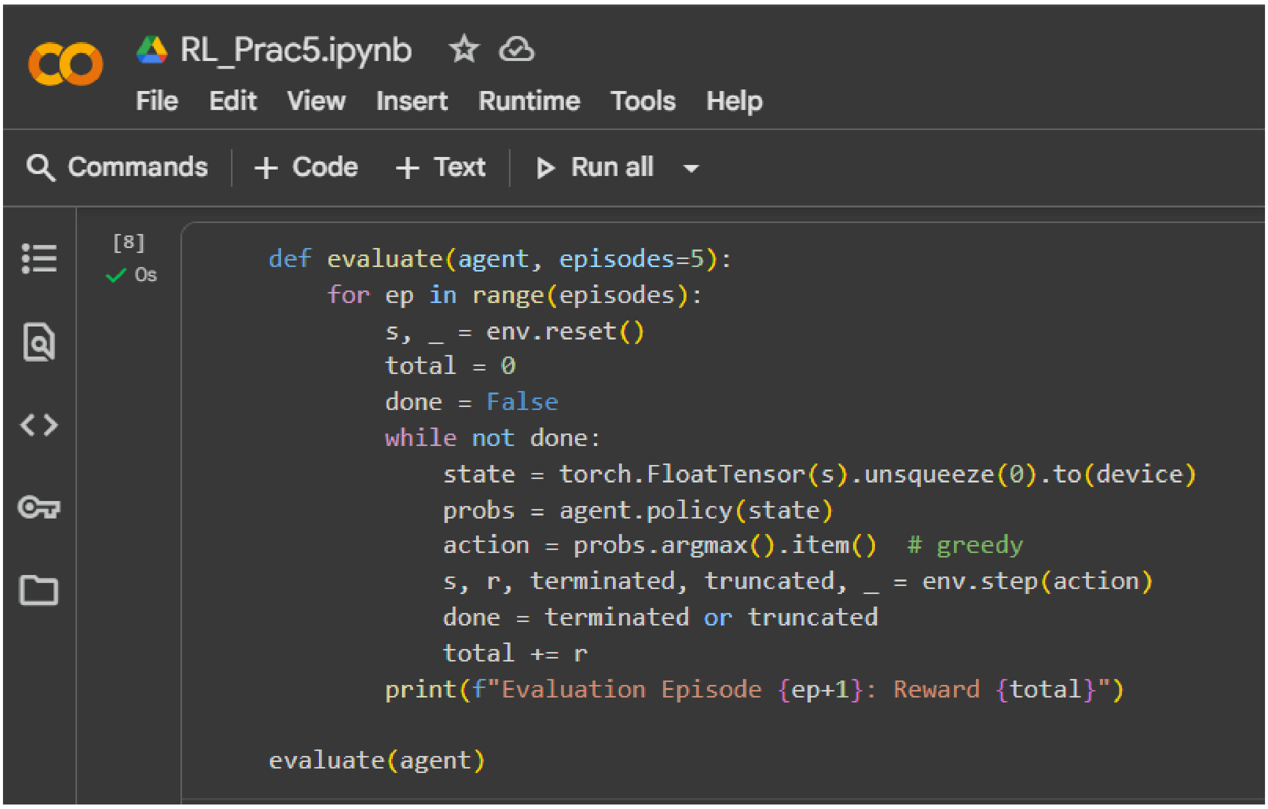
```
class REINFORCE:
    def __init__(self, state_dim, action_dim, lr=1e-2, gamma=0.99):
        self.policy = PolicyNetwork(state_dim, action_dim).to(device)
        self.optimizer = optim.Adam(self.policy.parameters(), lr=lr)
        self.gamma = gamma

    def select_action(self, state):
        state = torch.FloatTensor(state).unsqueeze(0).to(device)
        probs = self.policy(state)
        dist = torch.distributions.Categorical(probs)
        action = dist.sample()
        return action.item(), dist.log_prob(action)

    def update(self, rewards, log_probs):
        # Compute discounted returns
        returns = []
        G = 0
        for r in reversed(rewards):
            G = r + self.gamma * G
            returns.insert(0, G)
        returns = torch.tensor(returns).float().to(device)
        returns = (returns - returns.mean()) / (returns.std() + 1e-9)  # normalize

        loss = []
        for log_prob, Gt in zip(log_probs, returns):
            loss.append(-log_prob * Gt)
        loss = torch.stack(loss).sum()

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```



```
env = gym.make("CartPole-v1")

state_dim = env.observation_space.shape[0]
action_dim = env.action_space.n
agent = REINFORCE(state_dim, action_dim)

MAX_EPISODES = 500
MAX_STEPS = 500
rewards = []

for ep in range(1, MAX_EPISODES+1):
    state, _ = env.reset()
    log_probs = []
    ep_rewards = []

    for t in range(MAX_STEPS):
        action, log_prob = agent.select_action(state)
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated

        log_probs.append(log_prob)
        ep_rewards.append(reward)
        state = next_state

        if done:
            break

    agent.update(ep_rewards, log_probs)
    total_reward = sum(ep_rewards)
    rewards.append(total_reward)
    print(f"Episode {ep}, Reward: {total_reward}")
```

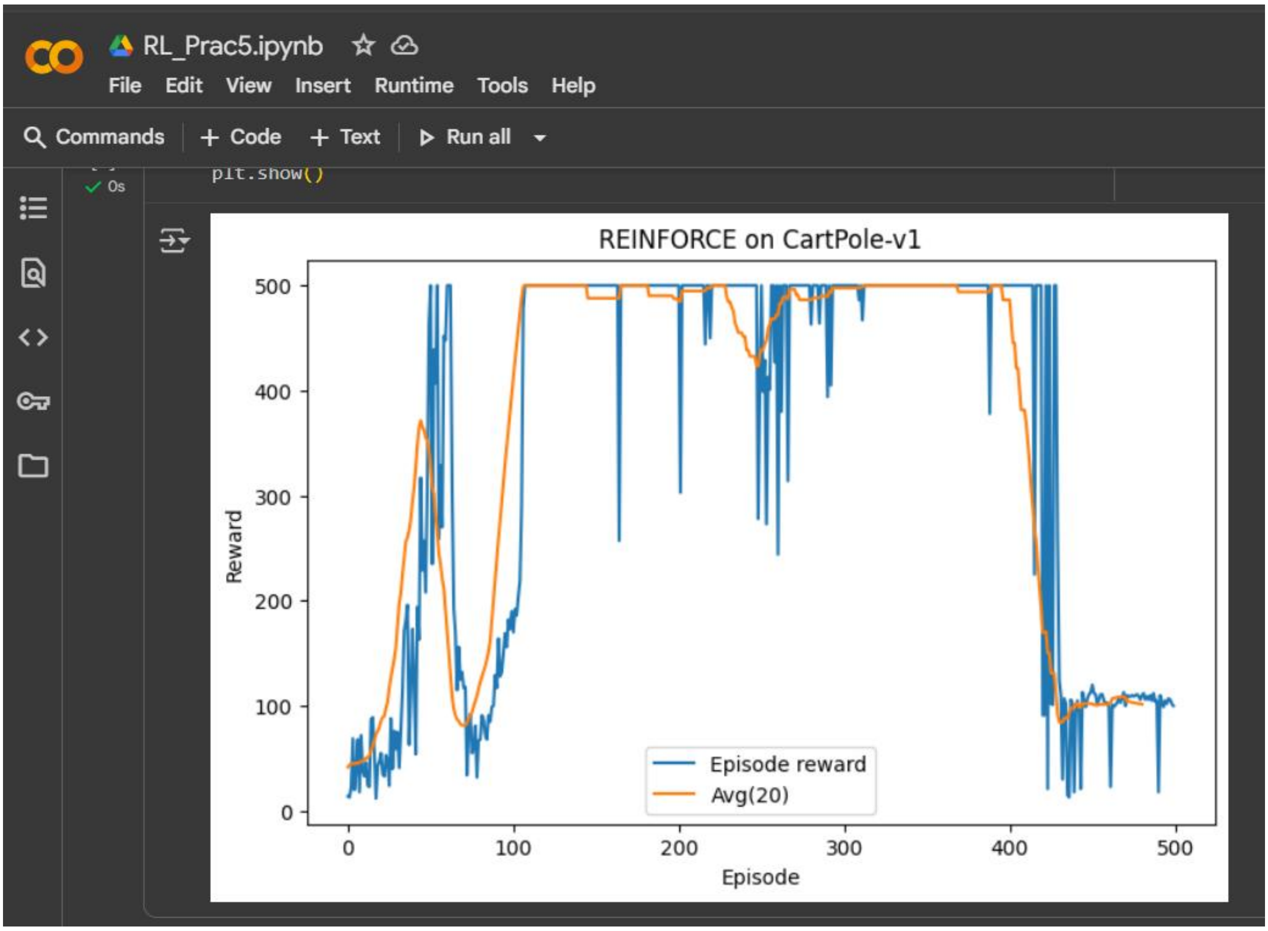


```
plt.figure(figsize=(8,5))
plt.plot(rewards, label="Episode reward")
plt.plot(np.convolve(rewards, np.ones(20)/20, mode='valid'), label="Avg(20)")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.title("REINFORCE on CartPole-v1")
plt.legend()
plt.show()
```

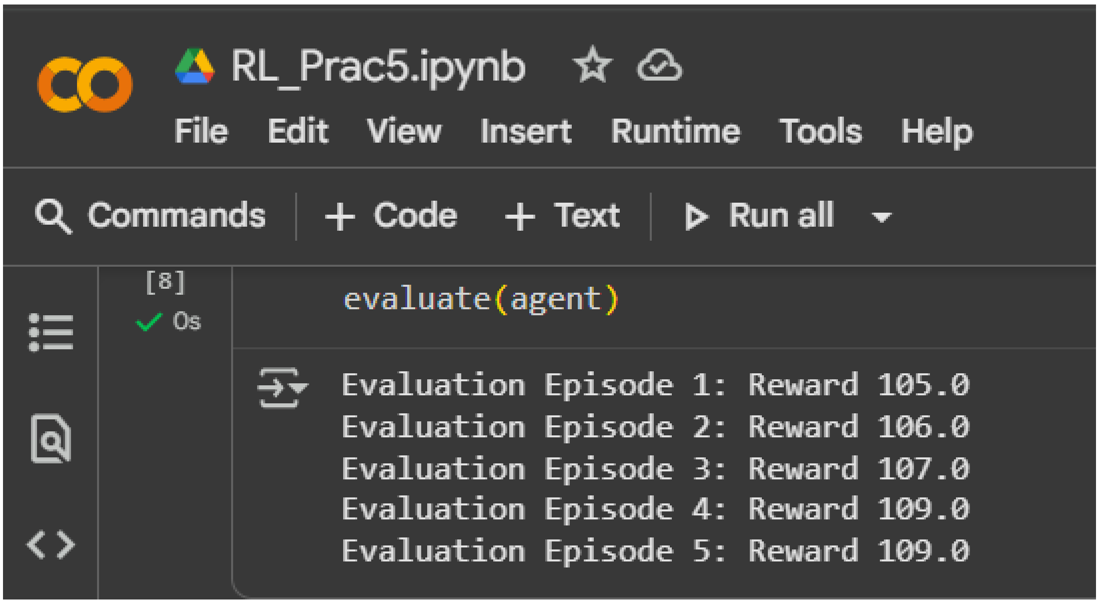

RL_Prac5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[8]
0s

```
def evaluate(agent, episodes=5):
    for ep in range(episodes):
        s, _ = env.reset()
        total = 0
        done = False
        while not done:
            state = torch.FloatTensor(s).unsqueeze(0).to(device)
            probs = agent.policy(state)
            action = probs.argmax().item()  # greedy
            s, r, terminated, truncated, _ = env.step(action)
            done = terminated or truncated
            total += r
        print(f"Evaluation Episode {ep+1}: Reward {total}")

evaluate(agent)
```

Output:



```
plt.show()
```



```
evaluate(agent)
```

```
Evaluation Episode 1: Reward 105.0
Evaluation Episode 2: Reward 106.0
Evaluation Episode 3: Reward 107.0
Evaluation Episode 4: Reward 109.0
Evaluation Episode 5: Reward 109.0
```

Grade/Marks **Sign of Lab Teacher with Date**

(_____ / 10)