# Explanation of Tabular Q-learning Implementation

This document explains how Tabular Q-learning is implemented in your code.   
The algorithm follows the standard Q-learning approach but is tailored for the CliffWalking-v1 environment.

## 1. Initialize Q-table

The Q-table is initialized with zeros. Its shape corresponds to the number of states and the number of possible actions in the environment:

n\_states = env.observation\_space.n  
n\_actions = env.action\_space.n  
Q = np.zeros((n\_states, n\_actions))

## 2. Training Loop

For each episode, the environment is reset and the agent interacts with it until termination.

## 3. Action Selection (ε-greedy)

The agent chooses an action using the ε-greedy strategy: with probability ε it selects a random action (exploration), and otherwise it selects the action with the highest Q-value (exploitation).

if np.random.rand() < epsilon:  
 action = env.action\_space.sample()  
else:  
 action = np.argmax(Q[state])

## 4. Environment Step and Q-value Update

After taking an action, the agent observes the next state and reward. The Q-value is updated using the Temporal Difference (TD) learning rule:

old\_value = Q[state, action]  
best\_next = np.max(Q[next\_state])  
td\_target = reward + gamma \* best\_next \* (not done)  
td\_error = td\_target - old\_value  
Q[state, action] += alpha \* td\_error

## 5. Track Metrics

During training, the following metrics are stored for later visualization:  
- Episode reward (cumulative sum of rewards per episode)  
- Epsilon (exploration rate over time)  
- TD errors (average temporal-difference errors per episode)

## 6. Epsilon Decay

At the end of each episode, epsilon is decayed to gradually shift from exploration to exploitation:

epsilon = max(min\_epsilon, epsilon \* epsilon\_decay)