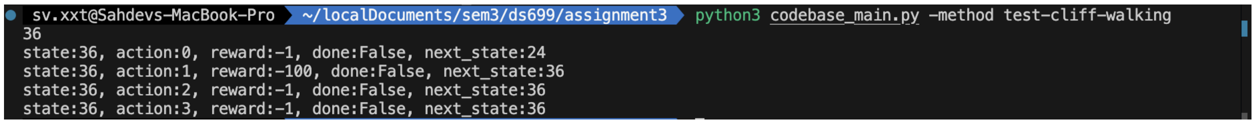
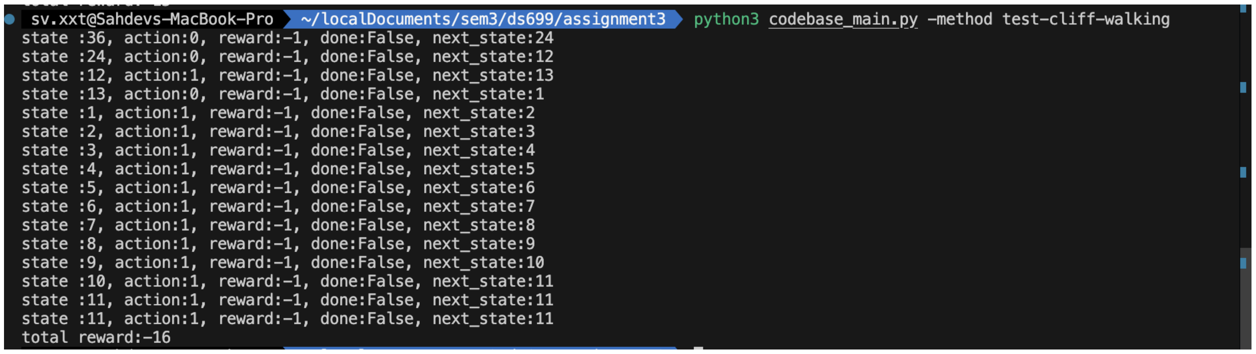
I a.

1. 

```
sv.xxt@Sahdevs-MacBook-Pro  ~/localDocuments/sem3/ds699/assignment3  python3 codebase_main.py -method test-cliff-walking
36
state:36, action:0, reward:-1, done:False, next_state:24
state:36, action:1, reward:-100, done:False, next_state:36
state:36, action:2, reward:-1, done:False, next_state:36
state:36, action:3, reward:-1, done:False, next_state:36
```

2. The initial state number is 36. It is right before the cliff actually begins and its coordinates are [3,0].
3. 0 - move up, 1 - move right, 2 - move down, 3 - move left.
4. 0, 11, 47 (terminal state) are at the corners of the state space. The state number is given by $rowNumber * numColumns + colNum$, where both rowNumber and colNumber start from 0. colNums is 12.
5. The agents steps into state 37, the cliff, incurring a reward of -100. It is not the terminal state.
6. The agent stays in the same state, 36, receiving a reward of -1 as it is bounded by the state space to not move beyond the edge.
7. The reward is -1 everywhere except for states 37 to 47, where it is -100. The transition probability is 1 for every move as it is non-stochastic.

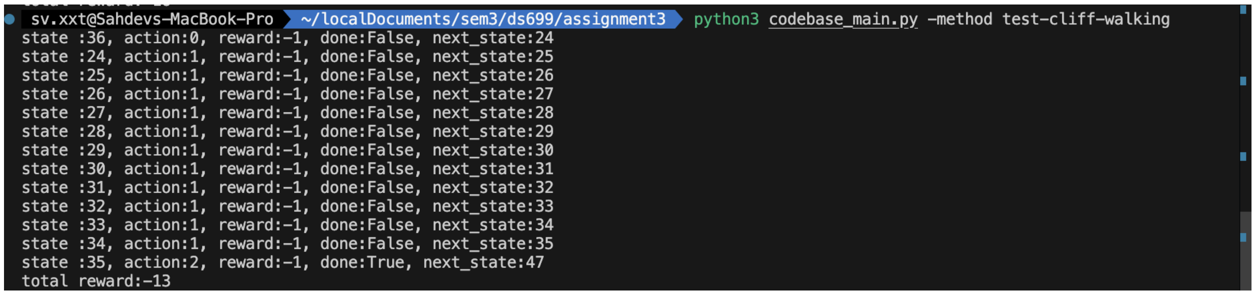
I b.

1. 

```
sv.xxt@Sahdevs-MacBook-Pro  ~/localDocuments/sem3/ds699/assignment3  python3 codebase_main.py -method test-cliff-walking
state :36, action:0, reward:-1, done:False, next_state:24
state :24, action:0, reward:-1, done:False, next_state:12
state :12, action:1, reward:-1, done:False, next_state:13
state :13, action:0, reward:-1, done:False, next_state:1
state :1, action:1, reward:-1, done:False, next_state:2
state :2, action:1, reward:-1, done:False, next_state:3
state :3, action:1, reward:-1, done:False, next_state:4
state :4, action:1, reward:-1, done:False, next_state:5
state :5, action:1, reward:-1, done:False, next_state:6
state :6, action:1, reward:-1, done:False, next_state:7
state :7, action:1, reward:-1, done:False, next_state:8
state :8, action:1, reward:-1, done:False, next_state:9
state :9, action:1, reward:-1, done:False, next_state:10
state :10, action:1, reward:-1, done:False, next_state:11
state :11, action:1, reward:-1, done:False, next_state:11
state :11, action:1, reward:-1, done:False, next_state:11
total reward:-16
```

2. The highest total reward for an episode is -13.

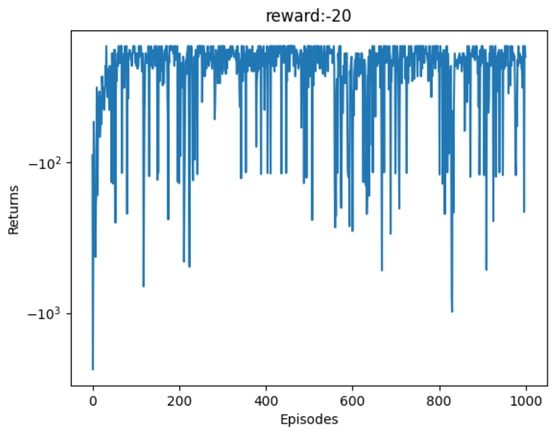
I c.

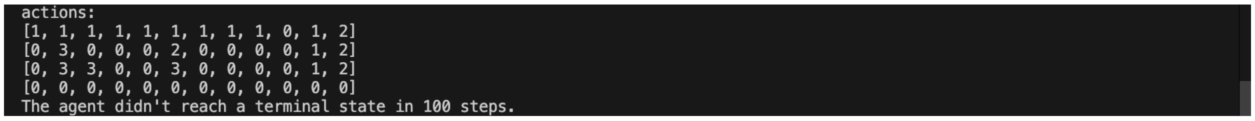
1. 

```
sv.xxt@Sahdevs-MacBook-Pro  ~/localDocuments/sem3/ds699/assignment3  python3 codebase_main.py -method test-cliff-walking
state :36, action:0, reward:-1, done:False, next_state:24
state :24, action:1, reward:-1, done:False, next_state:25
state :25, action:1, reward:-1, done:False, next_state:26
state :26, action:1, reward:-1, done:False, next_state:27
state :27, action:1, reward:-1, done:False, next_state:28
state :28, action:1, reward:-1, done:False, next_state:29
state :29, action:1, reward:-1, done:False, next_state:30
state :30, action:1, reward:-1, done:False, next_state:31
state :31, action:1, reward:-1, done:False, next_state:32
state :32, action:1, reward:-1, done:False, next_state:33
state :33, action:1, reward:-1, done:False, next_state:34
state :34, action:1, reward:-1, done:False, next_state:35
state :35, action:2, reward:-1, done:True, next_state:47
total reward:-13
```

P.T.O

II a.

1. 

2. 

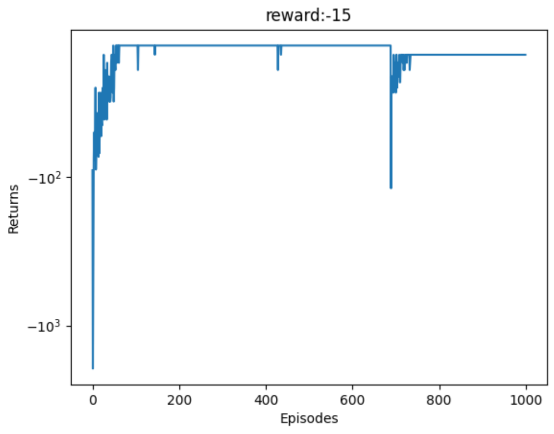
```
actions:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 2]
[0, 3, 0, 0, 0, 2, 0, 0, 0, 0, 1, 2]
[0, 3, 3, 0, 0, 3, 0, 0, 0, 0, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
The agent didn't reach a terminal state in 100 steps.
```

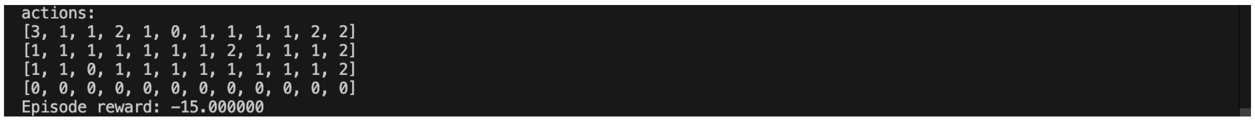
3. To be fair, it does signs of convergence in certain range of episodes but it isn't stable. This is because of a fixed epsilon that doesn't address the exploration vs exploitation problem in RL, explosion of rewards since there is no upper limit on the number of steps, improper initial states, etc.

4. Due to a fixed epsilon, the agent doesn't develop the ability to **adapt** between exploration and exploitation characteristics, leading to suboptimal convergence.

5. Initially, ϵ should be high, since action values are un-reliable initially, encouraging exploration by assigning a higher probability for it and it should reduce subsequently to focus on exploitation via greedy policy since the action values are more accurate later on.

II b.

1. 

2. 

```
actions:
[3, 1, 1, 2, 1, 0, 1, 1, 1, 1, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2]
[1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -15.000000
```

P.T.O

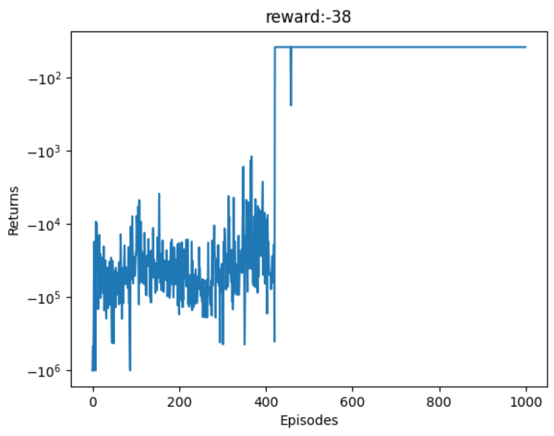
II c.

1. With num_steps = 10, the curve is initially rough and has plunges in later episodes whereas with 100 steps per update, the returns are less noisy and converges early.

2. The algorithm becomes equivalent to Monte-Carlo Control because with infinite steps, we essentially wait for the agent to reach the terminal state T.

$$G_t = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k}, \qquad Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)].$$

3. With large number of steps, the algorithm becomes slower, bias reduces and variance increases which is indicated by high fluctuations early on and then an almost unperturbed convergence. The following is the plot for $num\_steps = 100000$.



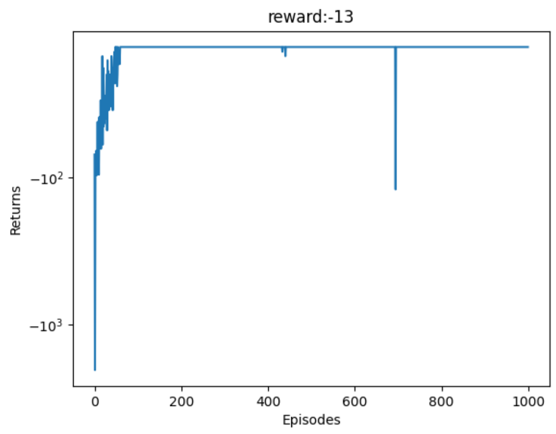
II d.

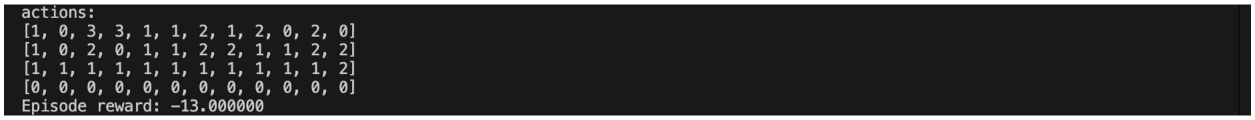
1. No, it is not the optimal policy. With negative (lower) neighboring state-action values, the agent prefers staying in the visited states even under the epsilon greedy policy as only these states have a higher q-value. When I experimented with an initial q value of -100, my agent was stuck at one state.

2. With a reward of -100 on cliff states, -1 otherwise and an optimal policy having a reward of -13, it's better to have q values initialized near 0 and slightly positive to avoid following loopy policies. Not too high to avoid loops again.

P.T.O

III a.

1. 



```
actions:
[1, 0, 3, 3, 1, 1, 2, 1, 2, 0, 2, 0]
[1, 0, 2, 0, 1, 1, 2, 2, 1, 1, 2, 2]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Episode reward: -13.000000
```

2. Q-learning converges faster because of its off-policy nature as it uses the action value of the best action at the next state under greedy policy. In a way, it accelerates the drive towards optimal q value.

3. A higher reward since it is at an accelerated state when compared to SARSA, it bootstraps quickly towards the greedy actions and exploits it.

4. Yes.

III b.

1. No, different alphas do not affect the total reward per episode as it only affect the rate at which we reach the optimal q value for the state. The policy remains the same.

2. The large one allows for faster convergence as it adds a higher percentage (α is between 0 and 1) of the error term.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)].$$

3. Since exploration is governed by $\epsilon/episodeNumber$, a new reward tends to influence the q values with magnitudes proportional to alpha again. This explains the fluctuations in $\alpha = 0.9$ whereas $\alpha = 0.1$ learns the optimal policy slowly and doesn't allow the q values to be influenced by a different reward after convergence.