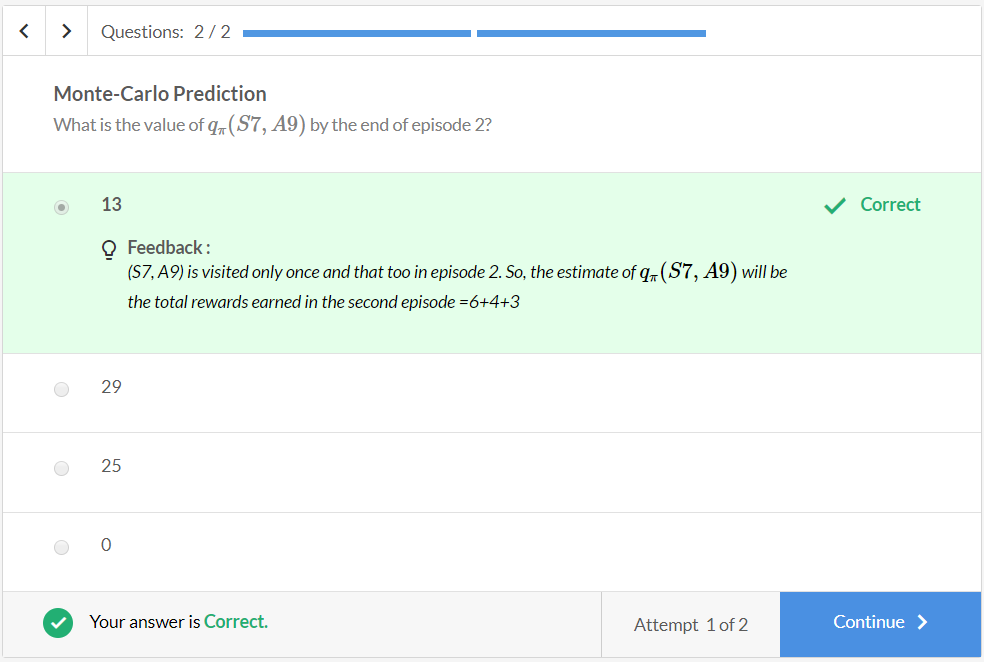
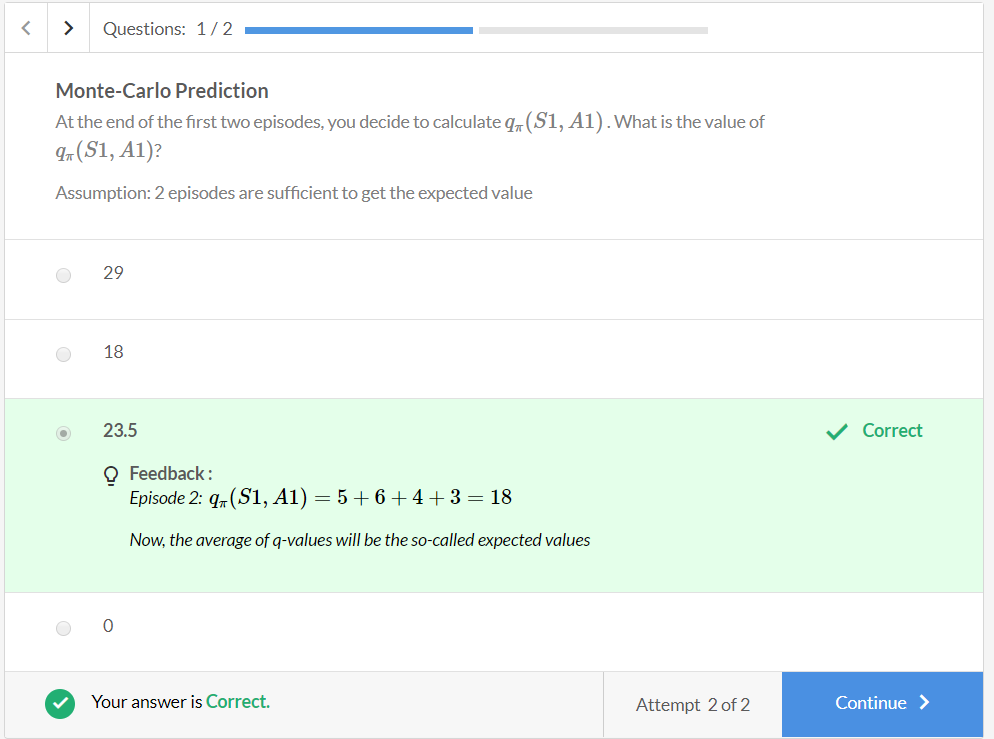
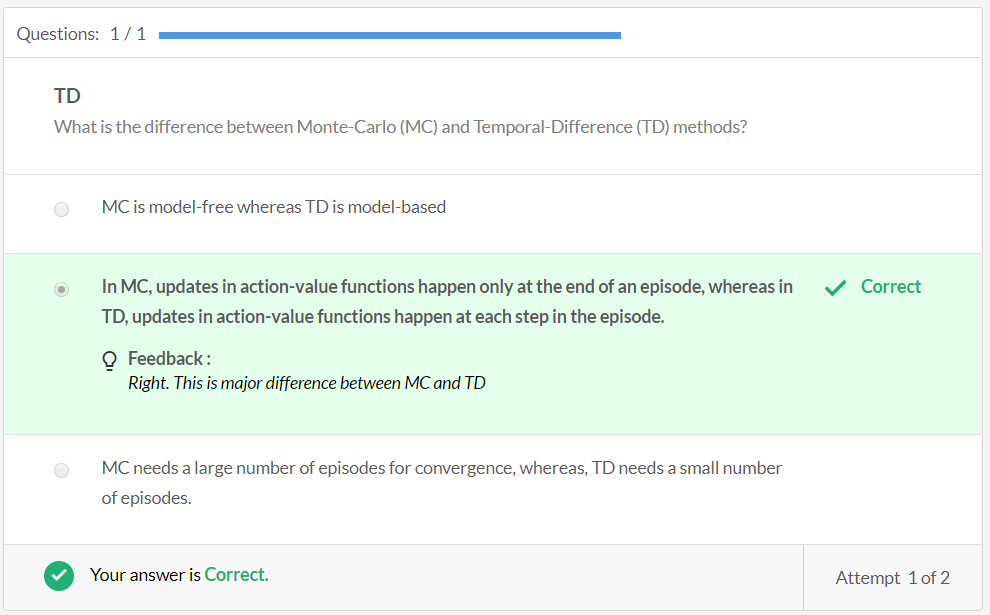
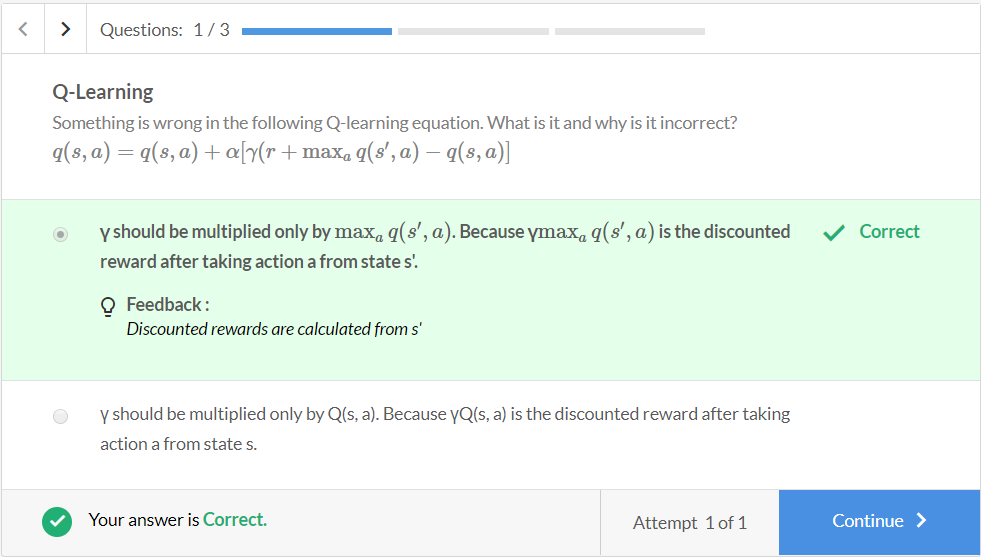
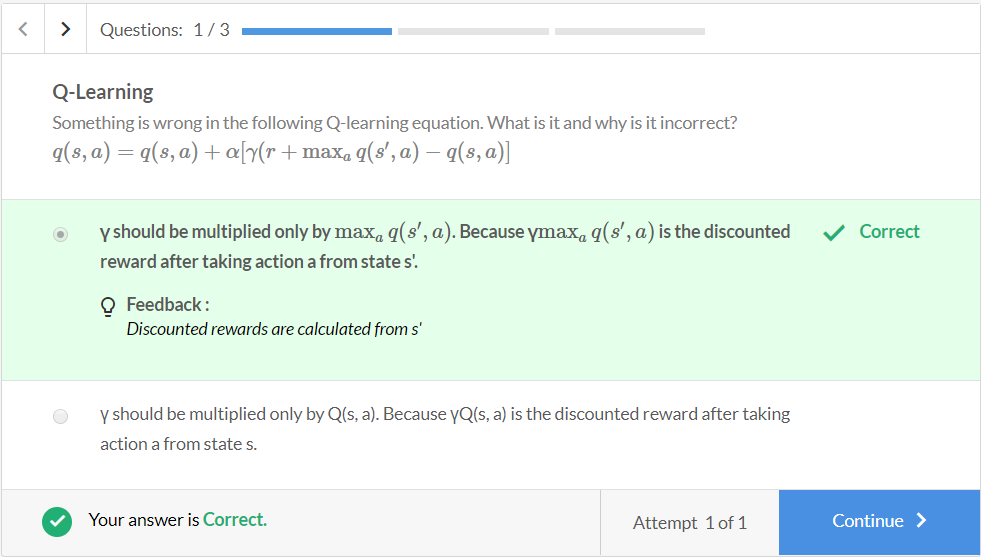


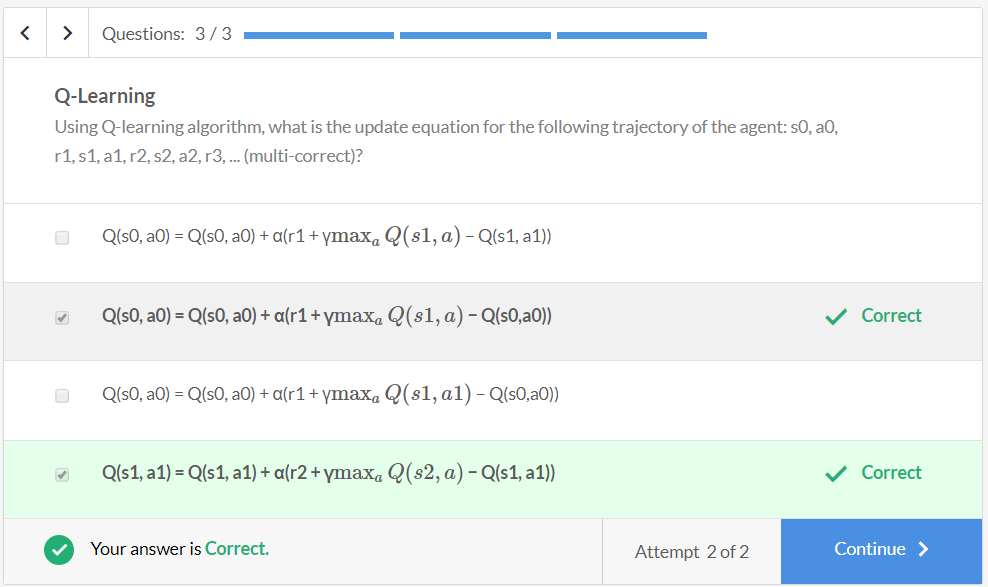
## Why is 'exploring starts' required? (Optional Reading)

To get an estimate of the state (s, a), you run multiple episodes starting from the same state-action pair (s, a), so you’ll need to start multiple episodes from (s, a), but what you need to ensure is that once the agent reaches s', it takes all possible actions from s' in various episodes (and not just take the ones that are more probable according to the current policy). Because if that happens, then all the ri  (the total rewards in various episodes) will basically be the same.



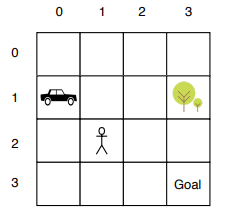






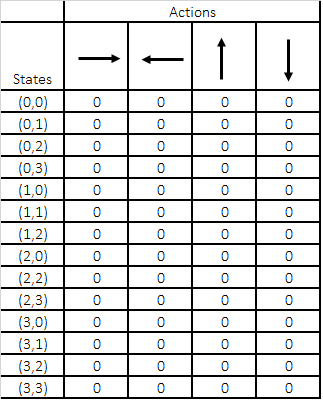
# Q-Learning Comprehension

In this segment, you'll use the same [GridWorld example](https://learn.upgrad.com/v/course/132/session/26081/segment/134626" \t "_blank).

[](https://learn.upgrad.com/course/117/module/7217/session/21600/segment/113373)

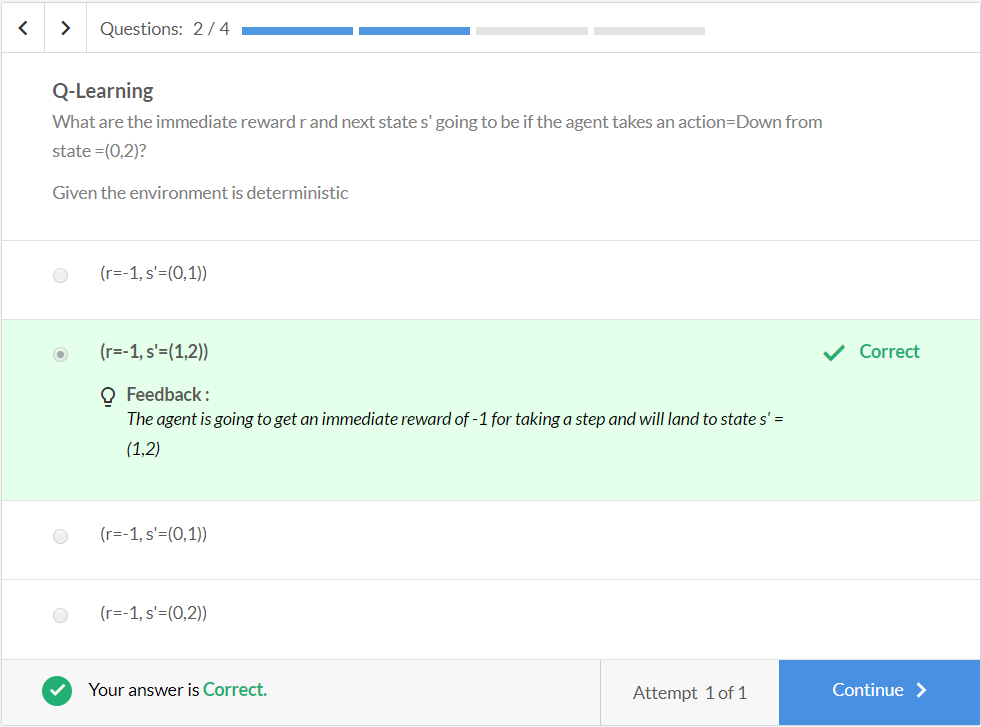
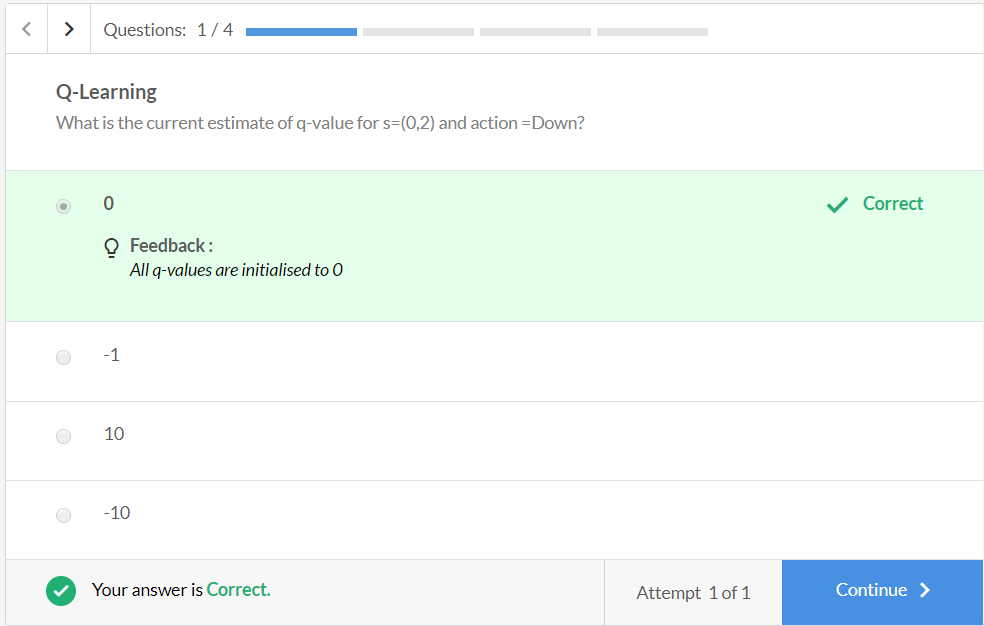
## Q-values Initialization:

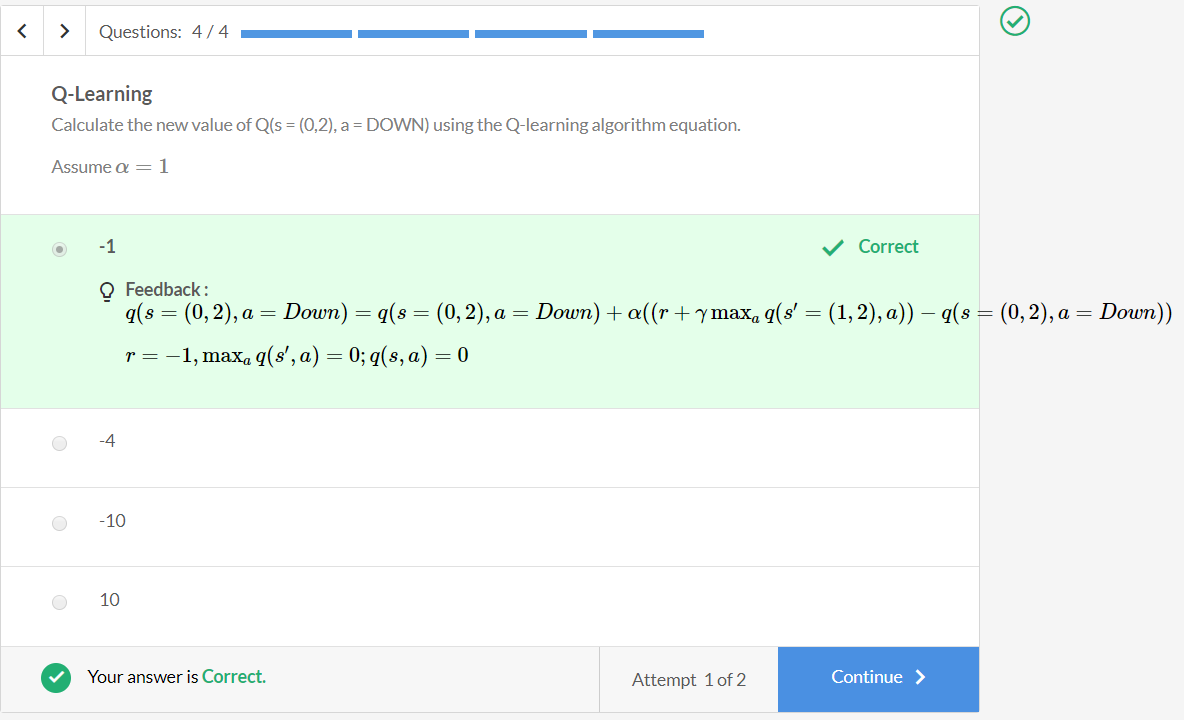
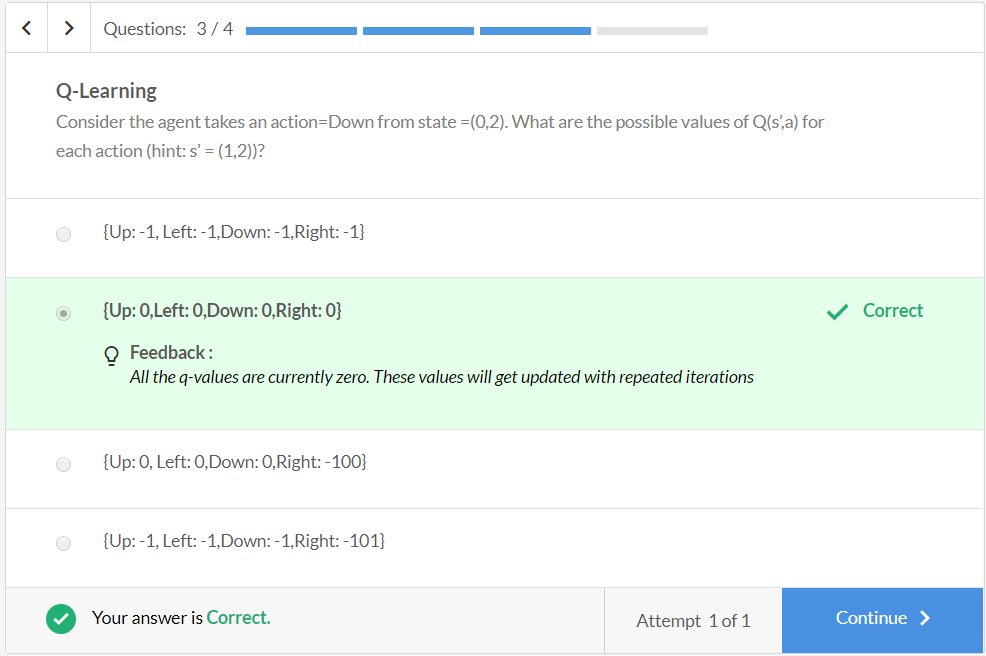
Say you initialise all Q-values as 0. We'll maintain a **Q-table**for storing q-values for each state-action pair.



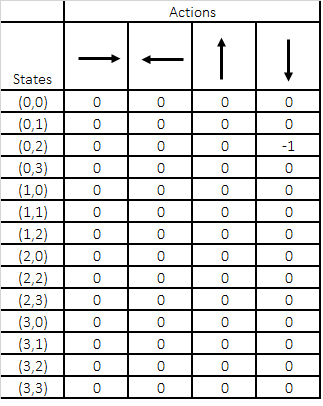
The locations of the obstacles and the goal are fixed, however, you randomly initialise the location of the agent at the beginning of each episode. Assume γ=1 .

In the first episode, say the agent randomly starts at s = (0, 2) and takes the action DOWN.





Now, you update your Q-table for s=(0, 2) and action=Down and move to the next state (1,2). So, the Q-table will look like the following:



Questions:1/1

**Q-Learning**

After updating the value of Q(s = (0,2), a = DOWN), the agent reaches s = (1,2). Out of the 4 possible actions, let’s consider the following:

Say the agent takes action RIght. Calculate the new value of Q(s = (1,2), a = Right) using the Q-update eqn, calculate the value of Q(s = (1,2), a = Right)

Assume α=1

Top of Form



**0**

**Feedback :**

q(s=(1,2),a=Right)=q(s=(1,2),a=Right)+α((r+γmaxaq(s′=(1,3),a))−q(s=(1,2),a=Right))

**Incorrect**



**-10**

**Feedback :**

q(s=(1,2),a=Right)=q(s=(1,2),a=Right)+α((r+γmaxaq(s′=(1,3),a))−q(s=(1,2),a=Right))

*r=-10, and all current estimates are 0*

**Correct**



-1



-11

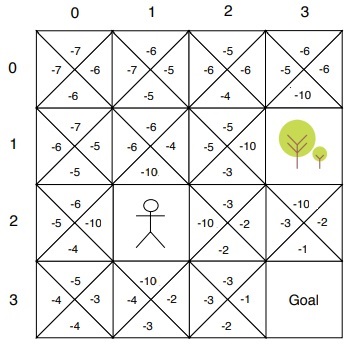
Bottom of Form

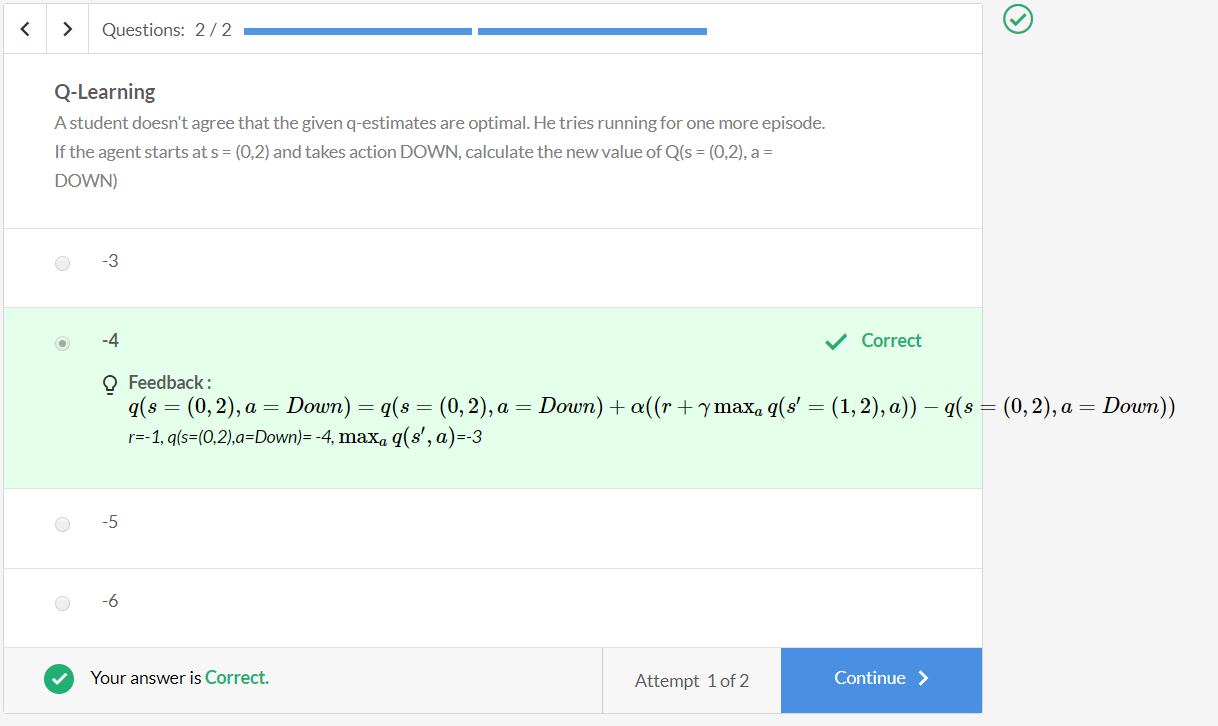
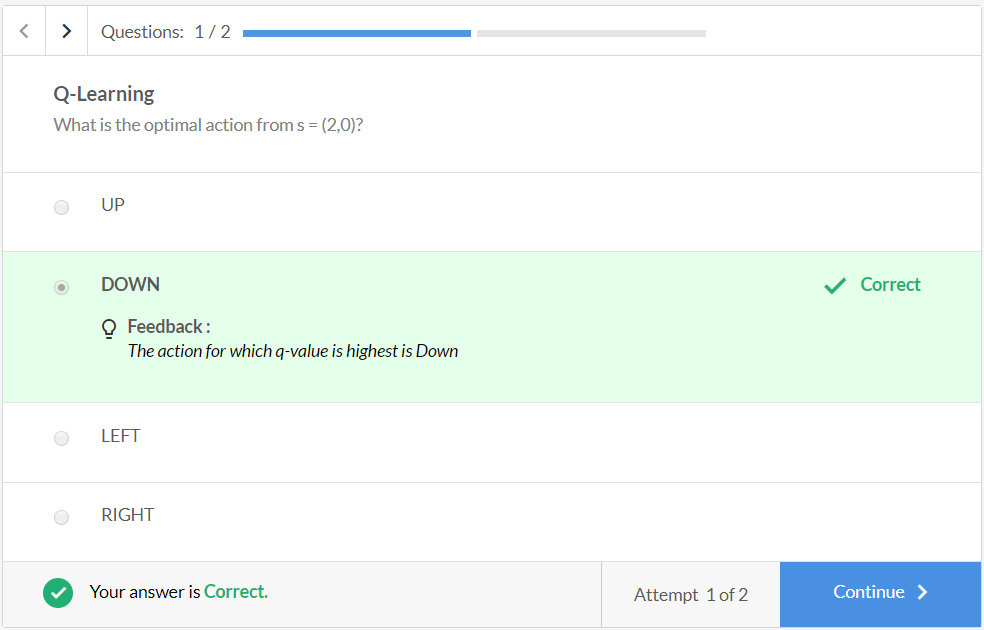
**close**Your answer is **Incorrect.**

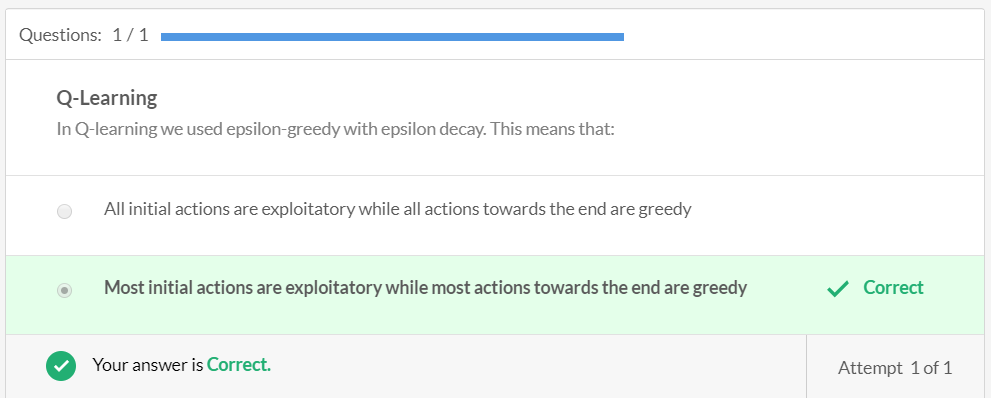
Attempt 2 of 2

You **update your Q-table** again and use those updated estimates to estimate the q-values again.

Say after a certain number of episodes, the optimal Q-values look like the ones in the figure below:



The q-values didn't change. This implies that the current estimates are optimal.



# Summary

In this session, you learnt two different kinds of model-free learning methods:

* **Monte-Carlo**
* **Temporal-difference (TD)**

You learnt how these can be applied to the reinforcement learning problem. Similar to DP, the problem was divided into a prediction problem and a control problem.

**Monte Carlo (MC)** methods learn value functions and optimal policies directly from the interaction with the environment. They average out the rewards earned from different episodes to estimate the action-value functions and use these action-value functions to improve the policy (ϵ -greedily).

ϵ is the hyperparameter that balances the trade-off between exploration and exploitation. Lower the value, lower is the exploration.

Then you learnt about **off-policy methods and importance sampling**. These methods deal with two kinds of policies: **behaviour policy and target policy.**And importance sampling ensures that important samples (important both for target and behaviour policy) are sampled more often than others.

Then, you learn about TD methods and in particular, **Q-Learning.** In TD methods, you can update the value after every time step. And, in practice, this update can be very advantageous when some of the states are extremely disastrous.

Q-learning directly **learns the optimal policy**, because the estimate of q-value is updated basis the estimate from the maximum estimate of possible next actions, regardless of which action you took.

You also learnt to write Q-learning and Monte-Carlo code in Python in the demonstration for Ad Placement Optimisation.

**Graded Questions**

