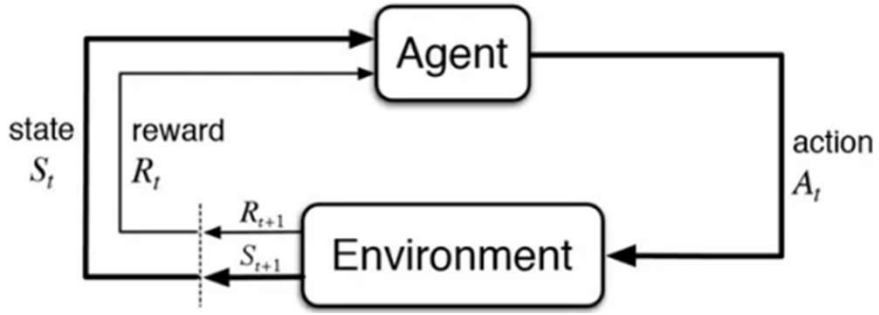
Reinforcement Learning

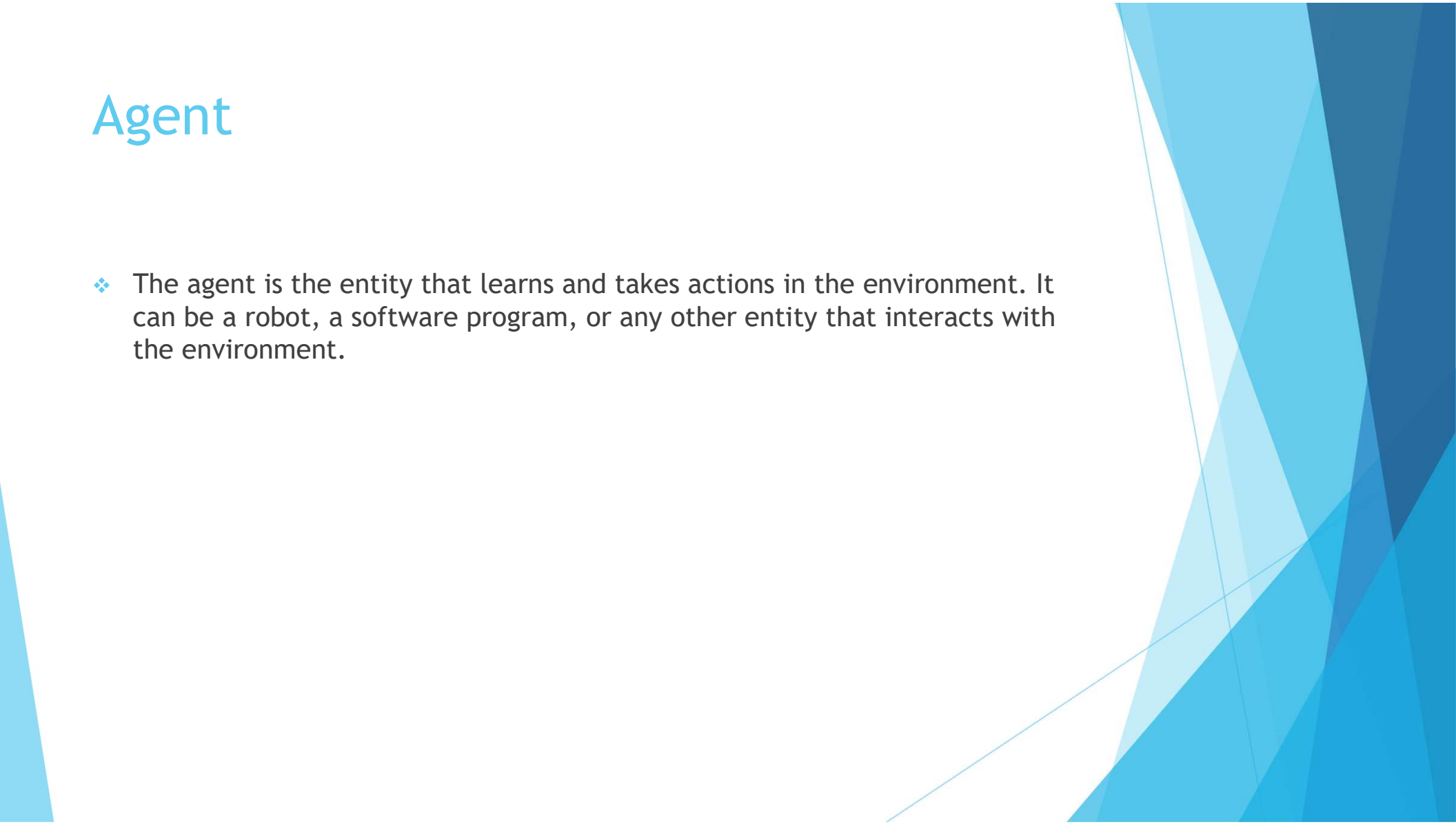
Reinforcement Learning



- Reinforcement Learning (RL) is a subfield of Artificial Intelligence that focuses on training an agent to make a sequence of decisions in an environment to maximize a cumulative reward.
- It has gained significant attention due to its ability to solve complex tasks by learning from interactions with the environment.

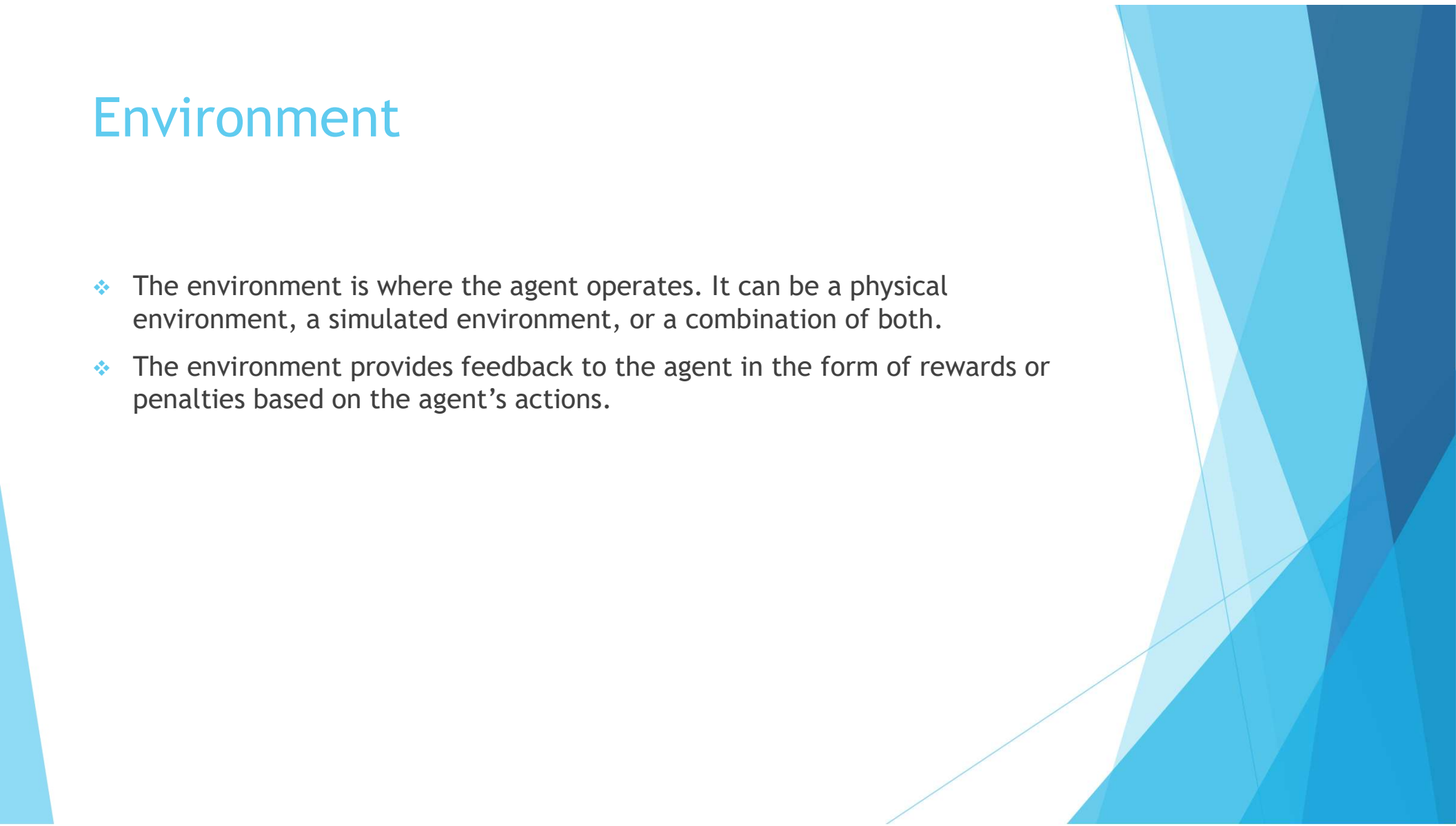
Agent

- The agent is the entity that learns and takes actions in the environment. It can be a robot, a software program, or any other entity that interacts with the environment.



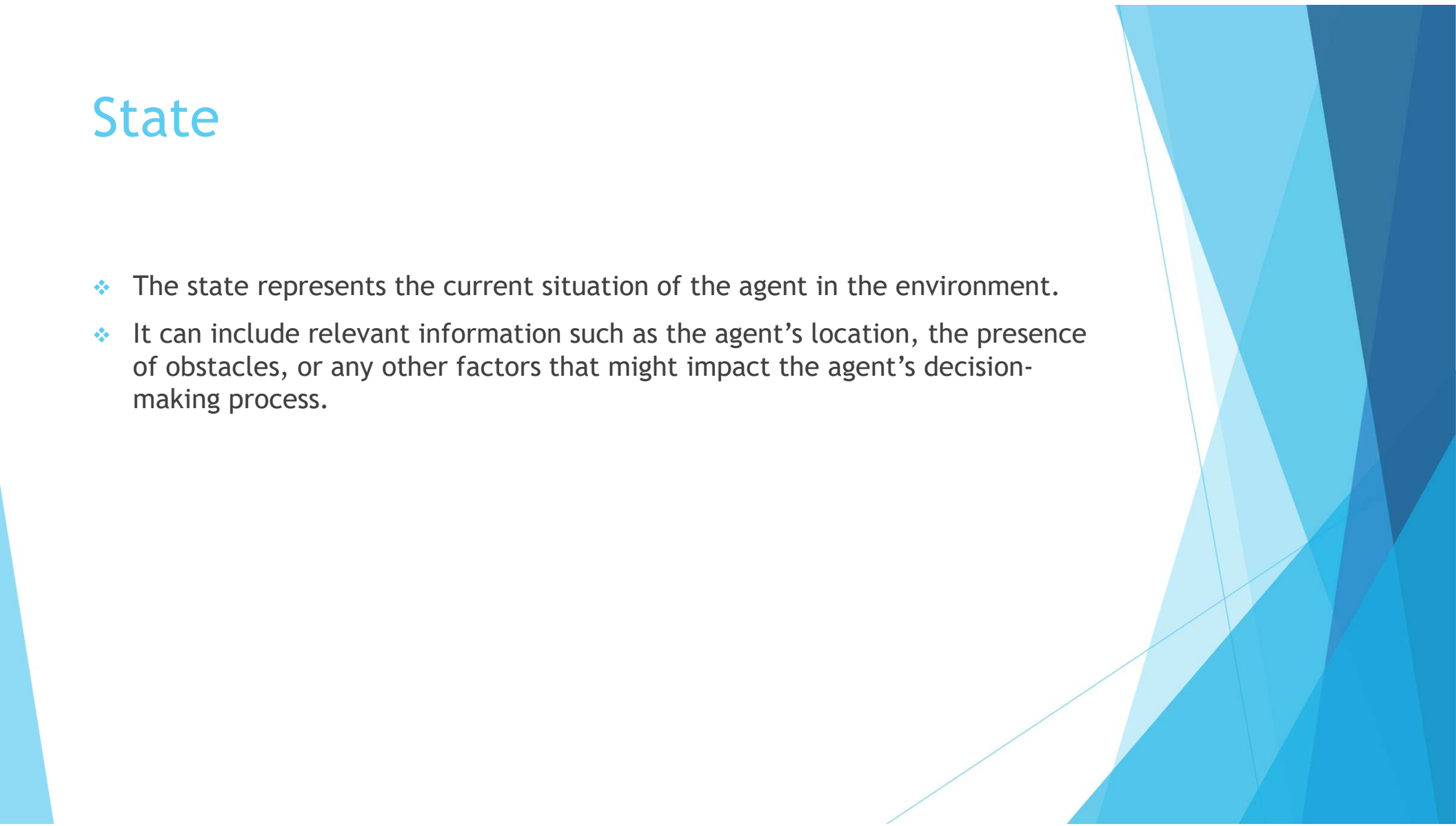
Environment

- The environment is where the agent operates. It can be a physical environment, a simulated environment, or a combination of both.
- The environment provides feedback to the agent in the form of rewards or penalties based on the agent's actions.



State

- The state represents the current situation of the agent in the environment.
- It can include relevant information such as the agent's location, the presence of obstacles, or any other factors that might impact the agent's decision-making process.



Action

- Decisions made by the agent based on the current state are referred to as actions.
- The agent selects an action from a set of possible actions, which then affects the state of the environment and potentially leads to rewards or penalties.

Reward

- Rewards are the positive or negative feedback that the agent receives from the environment based on its actions.
- The goal of reinforcement learning is to maximize the cumulative reward over time, i.e., to find an optimal policy that leads to the highest possible long-term reward.

Markov Decision Process (MDP)

- MDP is a mathematical framework to model decision-making in stochastic environments.
- It consists of a set of states, actions, transition probabilities, immediate rewards, and a discount factor.
- The agent interacts with the environment by selecting actions, and the environment transitions to a new state based on the probabilities.
- In an MDP, the Markov property holds, meaning the future state only depends on the current state and action, not the history.
- The goal is to find an optimal policy π that maximizes the expected cumulative reward.

Q-Learning

- Q-learning is a model-free reinforcement learning algorithm that learns the optimal action-value function, called the Q-function.
- The Q-function denotes the expected cumulative reward for taking a specific action in a given state.
- The algorithm updates the Q values iteratively based on the Bellman equation.

DQN (Deep Q-Network)

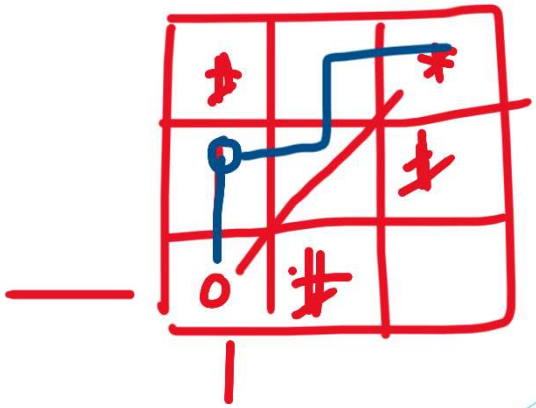
- DQN is an extension of Q-learning that uses deep neural networks to approximate the Q function.
- Instead of a lookup table for Q-values, a neural network is trained to predict the Q-values based on the current state.
- The network is trained using a combination of experience replay and a target network.
- Experience replay stores past experiences in a replay buffer, randomly sampling them for training to break correlations in the data.
- The target network is a separate network with delayed updates, providing stable Q-value targets during training.

Gym

- Gymnasium is a project that provides an API for all single agent reinforcement learning environments, with implementations of common environments: cartpole, pendulum, mountain-car, mujoco, atari, and more.

Markov's Decision Process: An Example

- Imagine a robot trying to navigate through a 3x3 grid to reach a goal in the top right corner.
- At each step, it can move up, down, left, or right.
- The robot has no prior knowledge of where it is and has to explore to figure out which actions will get it to the goal the fastest.



Markov's Decision Process: An Example

- **States**: The robot's position on the grid, such as (1,1) or (3,2).
- **Actions**: Move up, down, left, or right.
- **Transition Probabilities**: If the robot tries to move right, it may move right 80% of the time, but there's a 20% chance it moves in a random direction (due to a slippery floor).
- **Rewards**: The robot gets +10 points when it reaches the goal and -1 point for every wrong step it takes.
- **Policy**: After learning, the robot will develop a strategy (policy) that tells it which direction to move in every state to reach the goal as quickly as possible.

Markov's Decision Process

- An MDP provides a structured way to think about problems where an agent has to make decisions over time, in uncertain environments. By using the concepts of states, actions, rewards, and probabilities, the agent can learn an optimal strategy (policy) to achieve its goal, even when there's some randomness involved.
- In real life, MDPs are used in:
  - **Robotics**: Helping robots navigate environments.
  - **Gaming**: Teaching AI agents to play video games.
  - **Healthcare**: Creating treatment plans based on changing patient conditions.

Exploration and Exploitation Trade-off

- The **Exploration vs. Exploitation Tradeoff** is a key concept in **reinforcement learning** (RL) and decision-making. It refers to the dilemma an agent faces when choosing between two options:
  - **Exploitation**: The agent selects actions that it **already knows** will give the highest reward based on past experience.
  - **Exploration**: The agent tries **new actions** to discover if there are even better rewards that it hasn't yet encountered.
- **Why Is There a Tradeoff?**
  - If the agent **only exploits**, it might miss out on discovering a better action that could lead to higher long-term rewards.
  - If the agent **only explores**, it will waste time trying random actions, some of which may not be better, instead of taking advantage of what it already knows works.

Handling the Trade-off

- **Epsilon-Greedy Algorithm:**
  - The agent randomly explores with probability ε (epsilon), and exploits with probability 1 - ε.
  - For example, with ε = 0.1, the agent explores 10% of the time (choosing random actions) and exploits 90% of the time (choosing the best-known action).
- **Decaying Epsilon:**
  - Over time, the agent decreases ε, meaning it explores less as it gains more knowledge. Early on, it explores a lot, but as it learns, it focuses more on exploitation.

Rewards Signals and Returns

- A **reward signal** is the immediate feedback the agent receives after taking an action in a particular state. It's the environment's way of telling the agent how good or bad the action was, guiding the learning process.
  - **Immediate Feedback**: Rewards are given at each time step after an action is taken.
  - **Positive Reward**: Encourages the agent to repeat the action in similar situations. For example, if an agent playing a game collects points, those points are a positive reward.
  - **Negative Reward (or penalty)**: Discourages the agent from repeating the action. For example, if a robot crashes into a wall, it could receive a penalty (negative reward), indicating it should avoid such actions in the future.

Rewards Signals and Returns

- Imagine a robot navigating a grid to find a goal:
  - Moving closer to the goal might give a **+1 reward**.
  - Reaching the goal gives a **+10 reward**.
  - Hitting a wall or obstacle might give a **-1 penalty**.
- The robot receives these rewards right after taking an action, and they guide its behavior in future steps.

Rewards Signals and Returns

- **Returns** refer to the total accumulated reward over time from the current state onward. The agent aims to maximize the return, not just the immediate reward, because its actions may have long-term consequences.
- There are two main types of returns:
  - **Finite Horizon Return**: The total reward the agent expects to receive over a limited number of future steps.
  - **Infinite Horizon Return**: The total reward the agent expects to receive over an indefinite or infinite future.

Rewards Signals and Returns

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots = \sum_{k=0}^{T-t} R_{t+k+1}$$

- Where:
- G_t is the return at time step ttt.
- R_{t+1} is the reward received after taking an action at step ttt.
- T is the time step where the episode ends (infinite if there is no clear end).

Dynamic Programming and Policy Evaluation

- **Dynamic Programming (DP)** refers to a class of algorithms used to solve complex problems by breaking them down into simpler subproblems.
- In the context of **reinforcement learning (RL)**, DP provides a set of methods for solving **Markov Decision Processes (MDPs)** when the complete model of the environment (transition probabilities and rewards) is known.
- Dynamic programming methods are **offline** in the sense that they require a full model of the environment.
- They are not suited for environments where the transition probabilities and rewards are unknown (which is the case in most real-world problems), but they serve as foundational algorithms in RL.

Dynamic Programming and Policy Evaluation

- **Policy Evaluation** is the process of calculating the **value function** for a given policy π $V^{\pi}(s)$
- The value function tells us the expected return (cumulative future reward) starting from state **s** and following policy **π** thereafter.
- This process is central to understanding how good a particular policy is in an MDP.

Q-Learning(Off Policy)

- The core idea of Q-learning is to update the Q-value using the **Bellman Equation**. This equation expresses the Q-value of a state-action pair (s,a) as the immediate reward plus the maximum possible future reward the agent can receive from the next state.

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

- $Q(s,a)$ is the current Q-value for the state s and action a.
- α is the **learning rate** ($0 < \alpha \leq 1$), which controls how much new information overrides old information.
- r is the reward received for taking action a in state s.
- γ is the **discount factor** ($0 < \gamma \leq 1$), which determines the importance of future rewards.
- $\max_{a'} Q(s',a')$ is the maximum predicted reward for the next state s'.

Q-Learning Steps

- **Initialize**:
  - Initialize the Q-table arbitrarily (often with zeros) for all state-action pairs.
- **Loop for each episode**:
- Start in an initial state s.
  - **Choose an action** a using a policy, often **ε-greedy**:
    - With probability ε, choose a random action (explore).
    - With probability 1−ε1 choose the action with the highest Q-value (exploit).
  - **Take the action** a and observe the reward r and the new state s′.
  - **Update the Q-value** for the state-action pair (s,a) using the Q-value update rule.
  - **Move to the next state** s′ and repeat until the episode ends (i.e., the agent reaches a terminal state).
- **Repeat**:
  - Continue this process across many episodes until the Q-values converge and the agent learns an optimal policy.

Q – Learning Example

- Let’s consider a simple grid world where the agent needs to reach a goal from a starting position while avoiding obstacles. Here’s how Q-learning would work:
  1. The environment has states (positions on the grid) and actions (e.g., up, down, left, right).
  2. The agent starts at a random position.
  3. The agent takes an action (moves in one direction) and receives a reward based on the new position (e.g., +1 for the goal, -1 for hitting an obstacle).
  4. The agent updates its Q-value for the action it took, based on the reward and the maximum future Q-value from the new position.
  5. The process continues over many episodes until the agent learns to navigate the grid optimally.

Q – Learning Example

- Let's walk through a **simple Q-learning example** step by step, using a very basic environment to illustrate how the algorithm works. We'll use a small grid world where an agent learns to reach a goal.
  - The agent lives in a **5x5 grid**.
  - The agent starts in the bottom-left corner (state S_0) and needs to reach the top-right corn (goal state G).
  - Each action (move up, down, left, or right) results in a reward:
    - **+1** if the agent reaches the goal.
    - **-1** for hitting an obstacle (if applicable).
    - **0** for all other movements.
  - The episode ends when the agent reaches the goal.

Q - Learning Example

Actions Available:

- Up (U)
- Down (D)
- Left (L)
- Right (R)

Objective:

- The agent learns to navigate from the start to the goal by exploring and updating its Q-values using Q-learning.

Q - Learning Example

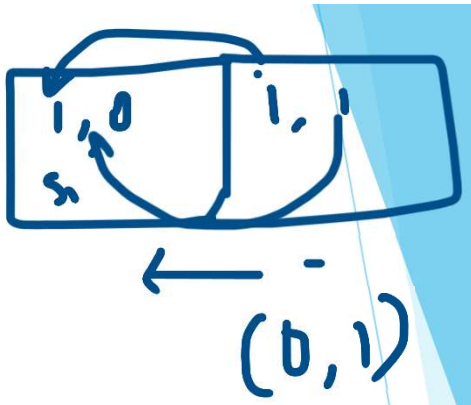
Q-Learning Algorithm Steps:

1. **Initialize** the Q-table with zeros for all state-action pairs.
2. **Repeat for each episode**:
   - Start in a random or designated initial state (e.g., $(4, 0)$).
   - Repeat for each time step in the episode:
     1. **Choose an action** using an **ε-greedy policy** (i.e., explore or exploit).
     2. **Take the action** and observe the **reward** r and the **next state** s'.
     3. **Update the Q-value** using the Q-learning update rule:

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

     4. Move to the new state s'.
   - End the episode when the agent reaches the goal state.

Q - Learning Example



Q-Table:

We represent each state-action pair in a **Q-table**, where:

- Rows represent states (grid positions).
- Columns represent actions (Up, Down, Left, Right).
- The value in each cell represents the current estimate of the total reward (expected return) for taking that action in that state.

State	Up	Down	Left	Right
(0, 0)	?	?	?	?
(0, 1)	?	?	?	?
...	...	...	...	...
(4, 4)	?	?	?	?

SARSA (On Policy)

- **SARSA** (State-Action-Reward-State-Action) is an on-policy reinforcement learning algorithm. It updates the Q-value based on the current state-action pair, the reward received, and the next state-action pair.
- Unlike Q-learning (which is off-policy), SARSA updates the Q-value using the actual action taken by the agent (following its policy), not the best possible future action. The update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$