Basic Concepts:

Agent: the learning and acting part of a Reinforcement Learning problem, which tries to maximize the rewards it is given by the Environment. Putting it simply, the Agent is the model which you are trying to design.

Actions: Actions are the Agent's methods which allow it to interact and change its environment, and thus transfer between states. Every action performed by the Agent yields a reward from the environment. The decision of which action to choose is made by the policy.

State: Every scenario the Agent encounters in the Environment is to formally called a state. The Agent transitions between different states by performing actions. It is also worth mentioning the terminal states, which mark the end of an episode. There are no possible states after a terminal state has been reached, and a new episode begins. Quite often, a terminal state is represented as a special state where all actions transition to the same terminal state with reward 0.

Policy (π): the policy, denoted as π, or sometimes as $\pi(\alpha, s)$, is a mapping from some state s to the probabilities of selecting each possible action given that state. For example, a greedy policy outputs for every state the action with the highest expected Q-value.

Reward: A numerical value received by the Agent from the Environment as a direct response to the Agent's actions. The Agent's goal is to maximize the overall reward it receives during an episode, and so rewards are the motivation the Agent needs in order to act in a desired behavior. All actions yield rewards, which can be roughly divided to three types: positive rewards which emphasize a desired action, negative rewards which emphasize and action the Agent should stray away from, and zero, which means the Agent didn't do anything special or unique.

Environment: Everything which isn't the Agent; everything the Agent can interact with, either directly or indirectly. The environment changes as the Agent performs actions; every such change is considered a state-transition. Every action the Agent performs yield a reward received by the Agent.

Value Function: Usually denoted as $V(s)$ (sometimes with a π subscript), the Value function is a measure of the overall expected reward assuming the Agent is in state s and then continuous playing until the end of the episode following some policy π. It is defined mathematically as:
$V(s) = E[\sum_{n=0}^{N} \gamma^n r_n]$

Episode: All states that come in between an initial-state and a terminal-state; for example: one game of Chess. The Agent's goal is to maximize the total reward it receives during an episode. In situations where there is no terminal-state, we consider an infinite episode. It is important to remember that different episodes are completely independent of one another.

Episodic Tasks: Reinforcement Learning tasks which are made of different episodes (meaning, each episode has a terminal state).

Continuous Tasks: Reinforcement Learning tasks which are not made of episodes, but rather last forever. This tasks have no terminal states. For simplicity, they are usually assumed to be made of one never-ending episode.

Expected Return: Sometimes referred to as "overall reward" and occasionally denoted as G, is the expected reward over an entire episode.

Discount Factor (γ): The discount factor, usually denoted as γ, is a factor multiplying the future expected reward, and varies on the range of [0. 1]. It controls the importance of the future rewards versus the immediate ones. The lower the discount factor is, the less import future rewards are, and the Agent will tend to focus on actions which will yield immediate rewards only.

More advanced concepts:

Monte Carlo (MC): Monte Carlo methods are algorithms which use repeated random sampling in order to achieve a result. They are used quite often in Reinforcement Learning algorithms to obtain expected values; for example - calculating a state Value function by returning to the same state

over and over again, and averaging over the actual cumulative reward received each time.

On-Policy & Off-Policy: Every Reinforcement Learning algorithm must follow some policy in order to decide which actions to perform at each state. Still, the learning procedure of the algorithm doesn't have to take into account that policy while learning. Algorithms which concern about the policy which yielded past state-action decisions are referred to as on-policy algorithms, while those ignoring it are known as off-policy.

Exploitation & Exploration: Reinforcement Learning tasks have no pre-generated training sets which they can learn from - they create their own experience and learn "on the fly". To be able to do so, the Agent needs to try many different actions in many different states in order to try and learn all available possibilities and find the path which will maximize its overall rewards; this is known as Exploration, as the Agent explores the Environment. On the other hand, if all the Agent will do is explore, it will never maximize the overall reward - it must also use the information it learned to do so. This is known as Exploitation, as the Agent exploits its knowledge to maximize the rewards it receives. The trade-off between the two is one of the greatest challenges of Reinforcement Learning problems, as the two must be balanced in order to allow the Agent to both explore the environment enough, but also exploit what it learned and repeat the most rewarding path it found.

Greedy Policy, ϵ-Greedy Policy: A greedy policy means the Agent constantly performs the action that is believed to yield the highest expected reward. Obviously, such a policy will not allow the Agent to explore at all. In order to still allow some exploration, an ϵ-greedy policy is often used instead; a number (named ϵ) in the range of [0, 1] is selected, and prior selecting an action, a random number in the range of [0, 1] is selected, and prior selecting an action, a random number in the range of [0, 1] is selected. If that number is larger than ϵ, the greedy action is selected - but if it's lower, a random action is selected. Note that if $\epsilon = 0$, the policy becomes the greedy policy, and if $\epsilon = 1$, always explore.

Bellman Equation: Formally, Bellman equation defines the relationships between a given state (or state-action pair) to its successors. While many forms exist, the most common one usually encountered in Reinforcement Learning tasks is the Bellman equation for the optimal Q-Value, which is given by: $Q^*(s,a) = \sum_{s',s} p(s',r|s,a)[r + \gamma, max_{a'} Q^*(s',a')]$ or when no uncertainty exists (meaning, probabilities are either 1 or 0): $Q^*(s,a) = r(s,a) + \gamma max_{a'} Q^*(s',a')$ where the asterisk sign indicates optimal value. Some algorithms, such as Q-Learning, are basing their learning procedure over it.

Q-Learning: Q-Learning is an off-policy Reinforcement Learning algorithm, considered as one of the very basic ones. In its most simplified form, it uses a table to store all Q-Values of all possible state-action pairs possible. It updates this table using the Bellman equation, while action selection is usually made with an ϵ-greedy policy.

Q Value (Q Function): Usually denoted as $Q(s,\alpha)$ (sometimes with a π subscript, and sometimes as $Q(s,\alpha;\theta)$ in Deep RL), Q Value is a measure of the overall expected reward assuming the Agent is in state s and performs action a, and then continues playing until the end of the episode following some policy π. Its name is an abbreviation of the word "Quality", and it is defined mathematically as: $Q(s,a) = [\sum_{n=0}^{N} \gamma^n r_n]$ where N is the number of states from state s till the terminal state, γ is the discount factor and r is the immediate reward received after performing action α in state s.