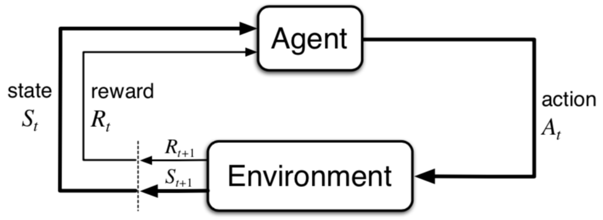
Reinforcement Learning Cheat Sheet

Recap

$\mathbb{E}[X] \doteq \sum_{x_i} x_i \cdot Pr\{X = x_i\}$
$\mathbb{E}[X|Y = y_j] = \sum_{x_i} x_i \cdot Pr\{X = x_i|Y = y_j\}$
$\mathbb{E}[X|Y = y_j] = \sum_{z_k} Pr\{Z = z_k|Y = y_j\} \cdot \mathbb{E}[X|Y = y_j, Z = z_k]$

Agent-Environment Interface



The Agent at each step t receives a representation of the environment's *state* $S_t \in \mathcal{S}$ and it selects an action $A_t \in \mathcal{A}(s)$. One time step later, as a consequence of its action, the agent receives a *reward*, $R_{t+1} \in \mathcal{R} \subseteq \mathbb{R}$ and goes to the new state S_{t+1}.
The MDP and agent together thereby give rise to a sequence or trajectory that begins like this:
$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 ...$

Return

The *return* is a some specific function of reward sequence. When there is a natural notion of final time step (T), the agent-environment interaction breaks naturally into sub-sequences (*episodes*). Each episodes ends in a special state called *terminal state*. $\mathcal{S}^+$ is the set of all states plus the terminal state.
The *total discounted return* is expressed as the sum of rewards (opportunely discounted with γ):

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad [3.8] \tag{1}$$
$$= R_{t+1} + \gamma G_{t+1} \quad [3.9] \tag{2}$$

Where γ is the *discount factor* and T is the final time step. It can be infinite. When there is a natural notion of final time step, we have the *episodes.*

Policy

A *policy* is a mapping from a state to probabilities of selecting each possible action:

$$\pi(a|s) \tag{3}$$

That is the probability of select an action $A_t = a$ if $S_t = s$.

Markov Decision Process

A finite **Markov Decision Process**, MDP, is defined by:
finite set of states: $s \in \mathcal{S}$,
finite set of actions: $a \in \mathcal{A}$
dynamics:

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \quad [3.2] \tag{4}$$

state transition probabilities:

$$p(s'|s, a) \doteq Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\}$$
$$= \sum_{r \in \mathcal{R}} p(s', r|s, a) \quad [3.4] \tag{5}$$

expected reward for state-action:

$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a]$$
$$= \sum_{r \in \mathcal{R}} r \cdot \sum_{s' \in \mathcal{S}} p(s', r|s, a) \quad [3.5] \tag{6}$$

expected reward for state-action-next state:

$$r(s', s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s']$$
$$= \sum_{r \in \mathcal{R}} r \cdot \frac{p(s', r|s, a)}{p(s'|s, a)} \quad [3.6] \tag{7}$$

Value Functions

State-Value function describes *how good* is to be in a specific state s under a certain policy π. Informally, is the expected return (expected cumulative discounted reward) when starting from s and following π

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] \quad [3.12] \tag{8}$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \quad [by\ 3.9] \tag{9}$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big] \quad [3.14] \tag{10}$$

The last one is the **Bellman equation for** v_π.
Action-Value function (Q-Function) describes *how good* is to perform a given action a in a given state s under a certain policy π. Informally, is the expected return when starting from s, taking action a and following π

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \quad [3.13] \tag{11}$$
$$= \sum_{s',r} p(s', r|s, a) \Big[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(a', s')\Big] \quad [Ex\ 3.17] \tag{12}$$

The last one is the **Bellman equation for** q_π.

Relation between Value Functions

$$v_\pi(s) = \sum_a \pi(a|s) \cdot q_\pi(s, a) \quad [Ex\ 3.12] \tag{13}$$
$$= \mathbb{E}_\pi[q_\pi(s, a)|S_t = s] \quad [Ex\ 3.18] \tag{14}$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big] \quad [Ex\ 3.13] \tag{15}$$
$$= \mathbb{E}\Big[R_{t+1} + \gamma v_\pi(s')|S_t = s, A_t = a\Big] \quad [Ex\ 3.19] \tag{16}$$

Optimal Value Functions

$$v_*(s) \doteq \max_\pi v_\pi(s) \quad [3.15] \tag{17}$$
$$= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \quad [3.18]$$
$$= \max_a \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_*(s')\Big] \quad [3.19]$$

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \quad [3.16] \tag{18}$$
$$= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r|s, a) \Big[r + \gamma \max_{a'} q_*(s', a')\Big] \quad [3.20]$$

$$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s, a) \tag{19}$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

Relation between Optimal Value Functions

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a) \Big[r + \gamma \sum_{a'} \pi(a'|s') q_*(s', a')\Big] \quad [Ex\ 3.25] \tag{20}$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_*(s')\Big] \quad [Ex\ 3.26] \tag{21}$$

Dynamic Programming

Collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a MDP.

Policy Evaluation [Prediction]

If the environment's dynamic is completely known we can use 10 solving the system of $|\mathcal{S}|$ equations in $|\mathcal{S}|$ unknowns $(v_\pi(s), s \in \mathcal{S})$.
We also can use an iterative solution:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \quad [4.5] \end{aligned} \tag{22}$$

We can compute new values $v_{k+1}(s)$ from old values $v_k(s)$ without change old values or update the values *in-place*.

Iterative Policy Evaluation for estimating $V \sim v_\pi$ (in-place version)

```
Inputs: $\pi$ - the policy to be evaluated
Params: $\theta$ - a small positive threshold determining the
  accuracy of the estimation
Initialize V(s), for all $s \in \mathcal{S}^+$ arbitrarily, except
  V(terminal) = 0
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ do
    foreach $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
end
```

Algorithm 1: Iterative Policy Evaluation - estimating $V \sim v_\pi$ - [§4.1]

Policy Iteration

Policy iteration consists of two simultaneous, interacting processes: one making the value function consistent with the current policy (*policy evaluation*), and the other making the policy greedy with respect to the current value function (*policy improvement*).

```
1. Initialization
Assign arbitrarily $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all
  $s \in S$,
2. Policy Evaluation
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ do
    foreach $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
end
3. Policy Improvement
policy-stable $\leftarrow$ true
foreach $s \in S$ do
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) \left[r + \gamma V(s')\right]$
    if old-action $\neq \pi(s)$ then
        policy-stable $\leftarrow$ false
    end
end
if policy-stable then
    return V $\approx v_*$ and $\pi \approx \pi_*$
else
    go to 2
end
```

Algorithm 2: Policy Iteration - estimating $\pi \sim \pi_*$ - [§4.3]

Value Iteration

Instead of waiting the convergence of $V(s)$ (policy evaluation loop) we can perform only one step of policy evaluation that, combined with policy improvement, lead to the following formulation:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \quad [4.10] \end{aligned} \tag{23}$$

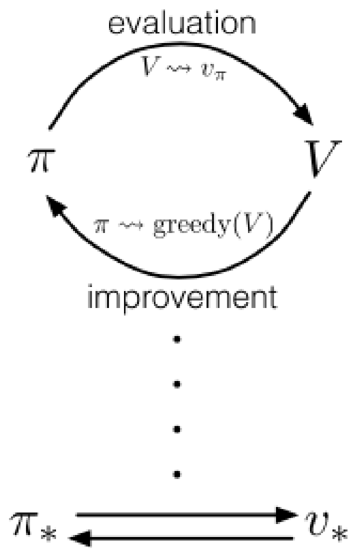
```
Params: $\theta$ - a small positive threshold determining the
  accuracy of the estimation
Initialize V(s), for all $s \in \mathcal{S}^+$ arbitrarily, except
  V(terminal) = 0
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ do
    foreach $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
end
output: Deterministic policy $\pi \approx \pi_*$ such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) \left[r + \gamma V(s')\right]$
```

Algorithm 3: Value Iteration - estimating $\pi \sim \pi_*$ - [§4.4]

One *sweep* is one update of each state.
In value iteration only a single iteration of policy evaluation is performed between each policy improvement. Value iteration combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.
Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep. The entire class of *truncated policy iteration* algorithms can be thought of as sequences of sweeps, some of which use policy evaluation updates and some of which use value iteration updates.

Generalized Policy Iteration

Generalized Policy Iteration is a way to refer to the general idea of letting policy-evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes.
Almost all reinforcement learning methods are well described as GPI. That is, all have identifiable policies and value functions, with the policy always being improved with respect to the value function and the value function always being driven toward the value function for the policy, as suggested by the diagram below [§4.6].


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π_*
v_*

Monte Carlo Methods

Monte Carlo (MC) methods require only experience from actual or simulated environment.

MC Prediction

```
Inputs: $\pi$ - the policy to be evaluated
Initialize: $V(s) \in \mathbb{R}$ for all $s \in \mathcal{S}$
Return(s) $\leftarrow$ an empty list for all $s \in \mathcal{S}$
while forever - for each episode do
    Generate an episode following $\pi$:
      $S_0, A_0, R_1, S_1, A_1, ..., S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    foreach step of episode, $t = T-1, T-2, ..., 0$ do
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t$ is not in the sequence $S_0, S_1, ..., S_{t-1}$
          (i.e. it is the first visit to $S_t$) then
            Append $G$ to Return($S_t$)
            $V(S_t) \leftarrow$ average(Return($S_t$))
        end
    end
end
```

Algorithm 4: On-policy First-visit Monte Carlo prediction - estimating $V \sim v_\pi$ [§5.1]

The *first-visit* is the first time a particular state has been observed.
The **first-visit MC** method estimates $v_\pi(s)$ as the average of the returns following first visits to s, whereas the **every-visit MC** method averages the returns following all visits to s. The every-visit MC Prediction is derived from first-visit version removing the "if" condition.
In other words we move backward from the step T and compute the G incrementally and associate the values of G to the current state and perform the average.

MC Estimation of Action Values

To determine a policy, if a model is not available, the state value is not sufficient and we have to estimate the values of state–action pairs.
The MC methods are essentially the same as just presented for state values, but now we have state–action pairs.
The only complication is that many state–action pairs may never be visited. We need to estimate the value of all the actions from each state, not just the one we currently favor. We can specify that the episodes start in a state–action pair, and that every pair has a nonzero probability of being selected as the start (assumption of *exploring starts*).

MC Control

```
Initialise: $\pi(s) \in \mathcal{A}(s)$ arbitrarily, for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily) for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
while forever do
    Choose $S_0 \in S$ and $A_0 \in A(S_0)$, randomly such
      that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$ following
      $\pi : S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$ $G \leftarrow 0$
      foreach step of episode, $t = T-1, T-2, ..., 0$ do
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t$ is not seen before, is not in the sequence
          $S_0, S_1, ..., S_{t-1}$ then
            Append $G$ to Returns($S_t, A_t$)
            $Q(S_t, A_t) \leftarrow$ average(Returns($S_t, A_t$))
            $\pi(S_t) \leftarrow \text{argmax}_a(Q(S_t, a))$
        end
    end
end
```

Algorithm 5: First-visit Monte Carlo (Exploring Starts) - estimating $\pi \sim \pi_*$ [§5.3]

To remove the exploring starts assumption, we can use an $\epsilon-$soft policy. Most of times it selects the greedy policy but with probability *epsilon* it instead selects an action at random.
Other approaches are the *off-policy* methods that learn about the optimal policy while behaving according to a different exploratory policy. The policy being learned about is called the *target policy*, π, and the policy used to generate behavior is called the *behavior policy*, b (usually an exploratory policy, e.g. random policy).
In order to use episodes from b to estimate values for *pi*, we require that every action taken under π is also taken, at least occasionally, under b. That is, we require that $\pi(a|s) > 0$ implies $b(a|s) > 0$. This is called the *assumption of coverage.*

Off-policy Every-visit MC Prediction

```
Inputs: $\pi$ - the policy to be evaluated
Initialize: $V(s) \in \mathbb{R}$ for all $s \in \mathcal{S}$
Return(s) $\leftarrow$ an empty list for all $s \in \mathcal{S}$
while forever - for each episode do
    Generate an episode following $b$:
      $S_0, A_0, R_1, S_1, A_1, ..., S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    foreach step of episode, $t = T-1, T-2, ..., 0$ do
        $G \leftarrow \gamma W G + R_{t+1}$
        Append $G$ to Return($S_t$)
        $V(S_t) \leftarrow$ average(Return($S_t$))
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
    end
end
```

Algorithm 6: Off-policy Every-visit Monte Carlo prediction - estimating $V \sim v_\pi$ [Course2-Week2]

Incremental Implementation

The average used to compute $V(S_t)$, can be performed incrementally:

$$V_n(S_t) = \frac{1}{n}\sum_{i=1}^{n} G_i(t) = V_{n-1}(S_t) + \frac{1}{n}(G_n(t) - V_{n-1}(S_t)) \tag{24}$$

Off-policy MC Control

The policy used to generate behavior, called the behavior policy, may in fact be unrelated to the policy that is evaluated and improved, called the target policy. An advantage of this separation is that the target policy may be deterministic (e.g., greedy), while the behavior policy can continue to sample all possible actions.

```
Initialize:for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Q(s,a) \in \mathbb{R}$ $C(s,a) \leftarrow 0$ $\pi(s) \leftarrow \text{argmax}_a + Q(s,a)$
while forever - for each episode do
    $b \leftarrow anysoftpolicy$ Generate an episode following
      $b$: $S_0, A_0, R_1, S_1, A_1, ..., S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    foreach step of episode, $t = T-1, T-2, ..., 0$ do
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow$
          $Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
        $\pi(s) \leftarrow \text{argmax}_a + Q(S_T, a)$ if $A_t \neq \pi(S_t)$
          then
            exit For Loop
        end
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
    end
end
```

Algorithm 7: Off-policy MC Control - estimating $\pi \sim \pi_*$ [§5.7]

Temporal-Difference Learning

TD Prediction

Starting from 24, we can consider a generic update rule of $V(S_t)$

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \quad [6.1] \tag{25}$$

α is a constant step-size and we call previous method *constant-α MC*. MC has to wait the end of an episode to determine the increment to $V(S_t)$, TD update the value at each step of the episode following the equation below:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \quad [6.2] \tag{26}$$

```
Inputs: $\pi$ - the policy to be evaluated
Params: step size $\alpha \in ]0,1]$
Initialize: $V(s) \in \mathbb{R}$ for all $s \in \mathcal{S}^+$ except for
  V(terminal)=0
foreach episode do
    Initialize $S$
    foreach step of episode - until $S$ is terminal do
        $A \leftarrow$ action given by $\pi$ for S
        take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$
        $S \leftarrow S'$
    end
end
```

Algorithm 8: Tabular TD(0) - estimating v_π [§6.1]

Recall that:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \quad [3.12|6.3] \quad (27)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \quad [by\ 3.9] \quad (28)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \quad [6.4] \quad (29)\end{aligned}$$

Monte Carlo methods use an estimate of 27 as a target. DP methods use an estimate of 29 as a target. The Monte Carlo target is an estimate because the expected value in 27 is not known; a sample return is used in place of the real expected return.
The DP target is an estimate because $v_\pi(S_{t+1})$ is not known and the current estimate, $V(S_{t+1})$, is used instead.
The TD target is an estimate for both reasons: it samples the expected values in 29 and it uses the current estimate V instead of the true v_π.
TD methods update their estimates based in part on other estimates. They learn a guess from a guess, i.e. they **bootstrap**. TD and MC methods have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions. The most obvious advantage of TD methods over Monte Carlo methods is that they are naturally implemented in an online, fully incremental fashion. With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need wait only one time step. In practice, TD methods have usually been found to converge faster than constant-α MC methods on stochastic tasks.

Sarsa

Sarsa (State-action-reward-state-action) is a on-policy TD control. Sarsa is sample-based version of policy iteration which uses Bellman equations for action values. The update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{30}$$

```
Params: step size $\alpha \in ]0,1]$, small $\epsilon > 0$
Initialise $Q(s,a)$ for all $s \in \mathcal{S}^+$ and $a \in \mathcal{A}(s)$,
  arbitrarily except that $Q(terminal - state, \cdot) = 0$
foreach episode do
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g.
      $\epsilon$-greedy)
    foreach step of episode - until $S$ is terminal do
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$
          (e.g. $\epsilon$-greedy)
        $Q(S,A) \leftarrow$
          $Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$
        $A \leftarrow A'$
    end
end
```

Algorithm 9: Sarsa - On-policy TD Control - estimating $Q \sim q_*$ [§6.4]

Q-Learning

Q-Learning is an off-policy TD control. Q-learning is a sample-based version of value iteration which iteratively applies the Bellman's optimality equation. The update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{31}$$

```
Params: step size $\alpha \in ]0,1]$, small $\epsilon > 0$
Initialize $Q(s,a)$ for all $s \in \mathcal{S}^+$ and $a \in \mathcal{A}(s)$,
  arbitrarily except that $Q(terminal - state, \cdot) = 0$
foreach episode do
    Initialize $S$
    foreach step of episode - until $S$ is terminal do
        Choose $A$ from $S$ using policy derived from $Q$
          (e.g. $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow$
          $Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    end
end
```

Algorithm 10: Q-Learning Off-policy TD Control - estimating $\pi \sim \pi_*$ [§6.5]

Expected Sarsa

Similar to Q-Learning, the update rule of Expected Sarsa, takes the expected value instead of the maximum over the next state:

$$\begin{aligned} &Q(S_t, A_t) \leftarrow \\ &Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) | S_{t+1}] - Q(S_t, A_t)] \\ &Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \quad [6.9\end{aligned}$$

The next action is sampled from π. However, the expectation over actions is computed independently of the action actually selected in the next state. In fact, it is not necessary that π is equal to the behavior policy. This means that Expected Sarsa, like Q-learning, can be used to learn off-policy without importance sampling.
If the target policy is greedy with respect to its action value estimates we obtain the Q-Learning. Hence Q-Learning is a special case of Expected Sarsa.

Planning and Learning

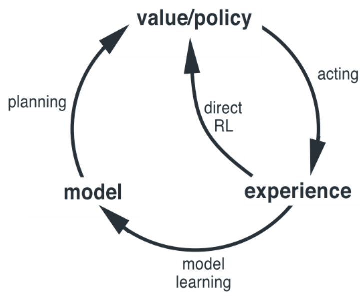
Planning uses simulated experience generated by a model, learning methods use real experience generated by the environment. Many ideas and algorithms can be transferred between planning and learning.

```
Params: step size $\alpha \in ]0,1]$, small $\epsilon > 0$
Initialize $Q(s,a)$ for all $s \in \mathcal{S}^+$ and $ain\mathcal{A}(s)$, arbitrarily
  except that $Q(terminal - state, \cdot) = 0$
foreach episode do
    1. Select a state $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(s)$,
       at random
    2. From a Sample Model obtain the sample reward
       and next state following: R, S' = model (S,A)
    3. Apply one-step tabular Q-Learning to
       $S, A, R, S'$:
    $Q(S,A) \leftarrow$
       $Q(S,A) + \alpha\,[R + \gamma \max_a Q(S',a) - Q(S,A)]$
end
```

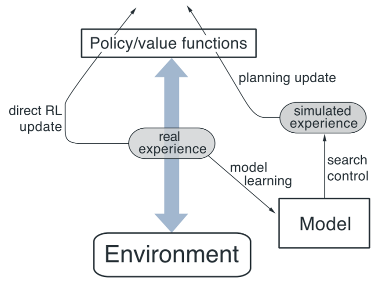
Algorithm 11: Random-sample one-step tabular Q-planning [§8.1]

Dyna

Within a planning agent, there are at least two roles for real experience:
it can be used to improve the model (to match more accurately the real environment), *model learning*,
it can be used to directly improve the value function and policy using the kinds reinforcement learning methods discussed before, *direct RL* [§8.2].



The experience can improve value functions and policies either directly or indirectly via the model (*indirect RL*). The real experience obtained with the interaction with the environment can be used to improve directly the Policy/value function (direct RL) or indirectly through the model learning and the planning that use simulated experience (*indirect RL*) [§8.2].

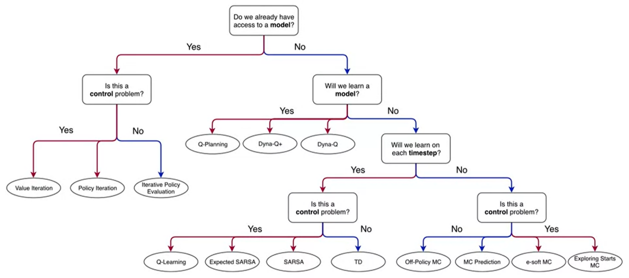


```
Initialize $Q(s,a)$ and $Model(S,A)$ for all $s \in \mathcal{S}^+$ and
  $a \in \mathcal{A}(s)$
while forever do
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \epsilon$-greedy$(S,Q)$
    (c) Take action $A$; observe resultant reward, $R$,
        and state, $S'$
    (d) $Q(S,A) \leftarrow$
        $Q(S,A) + \alpha\,[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic
        environment)
    (f) foreach n times do
        $S \leftarrow$ random previously observed state
        $A \leftarrow$ random action previously taken in $S$
        $R, S' \leftarrow Model(S,A)$
        $Q(S,A) \leftarrow$
          $Q(S,A) + \alpha\,[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    end
end
```

Algorithm 12: Dyna-Q [§8.2]

(d) Direct reinforcement learning
(e) Model-learning
(f) Planning

If (e) and (f) were omitted, the remaining algorithm would be one-step tabular Q-learning.
The agent responds instantly to the latest sensory information and yet always planning in the background. Also the model-learning process is in background. As new information is gained, the model is updated to better match reality. As the model changes, the ongoing planning process will gradually compute a different way of behaving to match the new model.
Models may be incorrect for many reasons: environment is stochastic and only a limited number of samples have been observed, the model was learned using function approximation that has generalized imperfectly, the environment has changed and its new behavior has not yet been observed. When the model is incorrect, the planning process is likely to compute a suboptimal policy. In some cases, the suboptimal policy computed by planning quickly leads to the discovery and correction of the modeling error. This happens when the model is optimistic, predicting greater reward or better state transition than are actually possible. It is more difficult to correct a model when the environment becomes better than it was before.
In Figure below there are represented the relation among algorithms presented in the Course on Coursera [Course3-Week1].


Do we already have access to a model?
Yes
No
Is this a control problem?
Will we learn a model?
Yes
No
Yes
No
Q-Planning
Dyna-Q+
Dyna-Q
Will we learn on each timestep?
Value Iteration
Policy Iteration
Iterative Policy Evaluation
Yes
No
Is this a control problem?
Is this a control problem?
Yes
No
No
Yes
Q-Learning
Expected SARSA
SARSA
TD
Off-Policy MC
MC Prediction
e-soft MC
Exploring Starts MC