PPO attepmts to solve the same problem as TRPO: get the biggest step size for policy update without breaking the policy. TRPO solves it with hard KL constaint and it uses second-order method (Hessian) to estimate the step size. PPO only involves first-order optimization. According to the published results, the performance of PPO is just as good as TRPO.
Two variants of PPO
PPO-Penalty solves the KL-contrained update like TRPO. It penalizes the KL-divergence in the objective instead of making it a hard constraint. The penalty coefficiently is adjusted automatically throughout the training.
PPO-Clip does not use KL-divergence to make the new policy close to the old one. It clips the surrgate gain (it clips the $\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)}$ term to $(1-\epsilon, 1+\epsilon)$ range to make sure the policy update is not too big.
PPO-Clip updates policies via

$$\theta_{k+1} = \text{argmax}_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}}[L(s, a, \theta_k, \theta)]$$

where

$$L(s, a, \theta_k, \theta) = \min(\frac{\pi_\theta(a|s)}{\pi_k(a|s)} A^{\pi_k}(s, a), \text{clip}(\frac{\pi_\theta(a|s)}{\pi_k(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_k}(s, a))$$

Interpretation: $\text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon)$ makes $\pi_\theta(a|s) \in [(1-\epsilon)\pi_\theta, (1+\epsilon)\pi_\theta]$
Suppose $A^{\pi_k}(a|s) < 0$, you want to make a less likely. The update objectively says $\pi_{k+1}(a|s) \geq (1-\epsilon)\pi_k(a|s)$; conversely, if $A^{\pi_k}(a|s)$ is positive, you want to make a more likely in your next policy iteration, the objective says $\pi_{k+1}(a|s) \leq (1+\epsilon)\pi_k(a|s)$.
The objective says the new policy can be at most ϵ "units" away from the old policy. This is some sort of trust-region optimization, ϵ defines the size of the trust region.

Algorithm 1: PPO-clip

1: Input: initial policy θ_0, initial value function ϕ_0
2: **for** $k = 0, 1, \cdots,$ **do**
3: Collect a set of trajectories $D_k = \{\tau_i\}$ by running policy π_k in the env
4: Compute rewards-to-go $\hat{R}_t$ (G)
5: Compute advantage estimate $\hat{A}_t$ based on the current value function ϕ_k
6: Update the policy by maximizing the PPO-Clip objective

$$\theta_{k+1} = \text{argmax}_\theta \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_k}(s_t, a_t)))$$

typically via Stochastic ascend with Adam
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \text{argmin}_\phi \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

typically via some gradient descent algorithm.

8: **end for**

How to get this algorithm into code?

1. Importance sampling interpretation of the policy loss allows us to use experiences from an old policy to update the current policy. The trajectory sampled from θ_k can be used by the algorithm to update π **a couple of times**. Each time of the update, we can compute $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_k}(a, s)$ and clip it to $[(1-\epsilon)A, (1+\epsilon)A]$ and compute the gradient of the loss with respect to the parameters of π_θ. This is somehow making the PPO a slightly-off-policy algorithm.
2. It might also be a good idea to clip the loss of the value function.
3. How do deep learning framework implement loss clipping? It is a node in a computation graph and it is not a differentiable operation? Same question goes to absolute value? I think they probably use some bump function that decreases sharply near the boundary points (clipping range). Same thing with absolute value, they probably have some way to smooth out the function at 0.
4. Good implementation of PPO uses parallel env to sample trajectory from many envs (see A2C).