Exercise 1: Discounted Rewards and Advantage Function (10 P)

You are given a list of tuples of the shape $\{(s_t, a_t, r_t)\}_{t=0}^{T}$ the length T of which you can infer from the length in the list. Implement the advantage estimation routine in Python for a discount factor γ as observed in on-policy policy gradient methods.
Given:

```
    states=[[s_t, a_t, r_t] for t in range(T)],
    value estimator v(s_t)
```

```
 discounted_rewards = []
cum_reward = 0.
for s_t, a_t, r_t in states[::-1]:  # go reverse from end of trajectory
    cum_reward = cum_reward + gamma * r_t
    discounted_rewards += [cum_reward]
# largest reward should be at beginning of trajectory
discounted_rewards = reversed(discounted_rewards)
# Compute A(s_t, a_t) = Q(s_t, a_t) - V(s_t)
A_t = []
for q_t, (s_t, _, _) in zip(discounted_rewards, states):
    A_t += [q_t - v(s_t)]
```

Exercise 2: DDPG Actor-Critic Loss (10 P)

Given a critic $Q_\phi(s, a)$ and an actor $\pi_\theta(a|s)$ write the PyTorch code to compute the gradients for both the critic and the actor, namely ∇_ϕ and ∇_θ.
Given:

```
    states=[[s_t, a_t, r_t, q_t] for t in range(T)],
    policy pi(s_t)
    critic critic(a_t, s_t),
    discount gamma
def actor _ critic _ loss (s_t, a_t, r_t, q_t, pi :  Callable, critic:  Callable):
      target_q_t = r_t + discount * next_q_t
    predict_next_q_t = critic(a_t, s_t)
    value_loss = (predict_next_q_t - target_q_t).pow(2).mean()
    actor_loss = - critic(pi(s_t), s_t).mean() # minus is important
     return value_loss, actor_loss
```