```
from cartpole import CartPoleEnv
import math
import numpy as np
import random

env = CartPoleEnv()
env.reset()

def discretize(val,bounds,n_states):
    discrete_val = 0
    if val <= bounds[0]:
        discrete_val = 0
    elif val >= bounds[1]:
        discrete_val = n_states-1
    else:
        discrete_val = int(round((n_states-1)*((val-bounds[0])/(bounds[1]-bounds[0]))))
    return discrete_val

def discretize_state(vals,s_bounds,n_s):
    discrete_vals = []
    for i in range(len(n_s)):
        discrete_vals.append(discretize(vals[i],s_bounds[i],n_s[i]))
    return np.array(discrete_vals,dtype=np.int)

# położenie, prędkość, kąt, prędkość kątowa
n_s = np.array([7,7,7,7])
n_a = env.action_space.n
#tablica zwierająca granice przedziałów
s_bounds = np.array(list(zip(env.observation_space.low,env.observation_space.high)))
s_bounds[1] = (-1.0,1.0)
s_bounds[3] = (-1.0,1.0)
#konieczna konwersja typu
s_bounds = np.dtype('float64').type(s_bounds)
```

```
def epsilon_greedy_action_from_Q(env, state, epsilon=0.2):
    if np.random.random() < epsilon:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q[tuple(state)])
    return action
```

```
Q = np.zeros(np.append(n_s,n_a))
Q=np.dtype('float64').type(Q)
print(Q.shape)
```

```
def Q_learing(env, episodes=1000, gamma=0.9, alpha=0.3):

    #tablica do której zapisujemy sumę nagród z każdego epizodu
    Rewards = []

    for i in range(episodes):

        if (i%100)==0:
            print("episode=",i)

        obs = env.reset()
        s = discretize_state(obs,s_bounds,n_s)

        episode_reward = 0
        finished = False



        time_step=0

        #zakończenie epizodu gdy 'finished == True' lub 'ilość kroków == 200'
        while not finished and not time_step==200:

            #DO UZUPEŁNIENIA
            a=epsilon_greedy_action_from_Q(env,s)
            obs, reward, finished, info = env.step(a)
            state_new = discretize_state(obs,s_bounds,n_s)

            indices = tuple(np.append(s,a))
            #next_indices = tuple(np.append(state_new,np.max(Q[state_new])))
            #action = np.max(Q[tuple(state)])
            #Q[indices] += alpha * (reward + gamma * np.max(Q[state_new]) - Q[indices])
            Q[indices] += alpha * (reward + gamma * np.max(Q[tuple(state_new)]) - Q[indice


           # Q[(s,a)] = Q[(s,a)] + alpha * (reward + gamma * np.max(Q[(state_new)]) - Q[(s
           # Q[s][a] = Q[s][a] + alpha * (reward + gamma * np.max(Q[state_new]) - Q[s][a])
            #Q[s[0], s[1], s[2], s[3], a] = Q[s[0], s[1], s[2], s[3], a] \
             #                                + alpha * (reward + gamma * (
              #      np.max(Q[state_new[0], state_new[1], state_new[2], state_new[3]])
              #      - Q[s[0], s[1], s[2], s[3], a]))
            #DO UZUPEŁNIENIA

            s = state_new


            #sumujemy wszystkie nagrody zdobyte w danym epizodzie
            episode_reward += reward
            time_step+=1

        Rewards.append(episode_reward)
```

```
            Rewards.append(episode_reward)

    return Q, Rewards


learning_episodes = 3000

_,R = Q_learing(env,learning_episodes)



#Wyliczamy średnią nagrodę - ilość epizodów (learning_episodes) dzielimy na 100
#i uśredniamy nagrody z kolejnych (learning_episodes/100) epizodów.
meanR= []
for i in range(100):
    meanR.append(np.mean(R[int(learning_episodes/100)*i:int(learning_episodes/100)*(i+1)]))


#wykres pokazuje nagrody zdobyte w 100 kolejnych epizodach wybranych z wszystkich epizodów
#oraz jak zmieniała się średnia nagroda zdobyta przez agenta

import matplotlib.pyplot as plt
x_data = range(0,100)
plt.plot(x_data,R[::int(learning_episodes/100)],label="reward")
plt.plot(x_data,meanR,label="mean reward")
plt.title('CartPole: Q-learing')
plt.xlabel('Episode')
plt.ylabel('Reward')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/gym/logger.py:30: UserWarning: WARN: Box bound
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
(7, 7, 7, 7, 2)
episode= 0
episode= 100
episode= 200
episode= 300
episode= 400
episode= 500
episode= 600
episode= 700
episode= 800
episode= 900
episode= 1000
episode= 1100
episode= 1200
episode= 1300
episode= 1400
episode= 1500
episode= 1600
episode= 1700
episode= 1800
episode= 1900


episode= 2200
episode= 2300
episode= 2400
episode= 2500
episode= 2600
episode= 2700
episode= 2800
episode= 2900
```