Collaboration policy for homework: This homework is to be done by yourself. Please do not collaborate with others, look for answers online, or post homework problems or solutions to websites or discord servers. You may discuss the homework concepts at a high level with other students. **We encourage you to come to office hours for help.**

Submit the following files to Canvas:

`hw4_solution.pdf` - PDF including your program output, and also answers questions in problems 2 and 3.

`taxi_sarsa.py`

`sarsa_qvals.pickle`

`sarsa_policy.pickle`

`taxi_qlearning.py`

`qlearning_qvals.pickle`

`qlearning_policy.pickle`

`multiagent_learning.py`

`report.pdf`

You may use any method to create your pdf that you want, it does not need to be Latex. However, you may not handwrite your answers, as we expect some graphs to be included.

Note: This homework asks you to write Python code "from scratch". What I mean by this is that I expect you to not import any advanced machine-learning libraries. However, I do expect you to use modules `pickle` for serialization of objects, and `matplotlib` (or something similar) to draw some fancy graphs.

Problem 1. (70 points)[Programming] For problem 1, you will be implementing two reinforcement learning algorithms to compute value functions and policies for the Taxi environment in OpenAI/Farama Gym. The problem in this environment is to pick up passengers in your taxi and drop them off at their destinations at which point the episode ends. It is similar to the Gridworld but with a few more features making up the states and more actions. There are 500 discrete states (25 taxi positions, 5 locations of passengers, and 4 destination locations). A state is represented by an int() calculated as `((taxi row * 5 + taxi col) * 5 + passenger location) * 4 + destination`. There are 6 discrete deterministic actions: move south, north, east, west, pickup passenger, and drop off passenger. The reward function consists of -1 for each non-terminal state (terminal states are when the location of the taxi with passenger is the same as the delivery location), +20 for delivering the passenger, and -10 for trying to pickup or drop off a passenger when there isn?t a passenger to be picked up or the wrong destination.

Please read https://gymnasium.farama.org/environments/toy_text/taxi/ for more details. A template of using the environment is in `hw4_taxi_env.py`, feel free to use this file but it is not required. (Note, you can use `env.render()` to visualize the environment.)

You will implement two algorithms: Sarsa and Q-learning and apply them to the Taxi environment specified above. Your code must take exactly three arguments: $\epsilon \in (0, 1)$ which is the exploration parameter, $\alpha \in (0, 1)$ which is the learning rate, $\gamma \in (0, 1)$ which is the discount factor. You must determine the optimal values for each of these three hyperparameters and the convergence criteria. (You are allowed to decay the exploration parameter ϵ if you?d like.) Your code must save three files which consist of: (1) action values after convergence, (2) policy after convergence, and (3) a figure of the cumulative reward per episode.

a. (35 points) Write Python code (from scratch) to implement the Sarsa algorithm in file `taxi_sarsa.py` and run your code on the Taxi environment to compute the optimal action-value function q^* and policy π^*. Your code must output three files:

   1. `sarsa_q_vals.pickle` which contains the dictionary `sarsa_q_vals` where the keys are the states (encoded exactly as given by the environment) and the values are dictionaries where the keys are the actions and the values are the learned q values $q(s, a)$,
   2. `sarsa_policy.pickle` which contains a dictionary sarsa policy where the keys are the states (encoded exactly as given by the environment) and the values are the policy actions (encoded as integers 0,1,...,5 exactly as the environment expects)
   3. a figure `sarsa_total_reward.png` which shows the total reward per episode (x-axis are the episodes and y-axis is the total reward per episode) in comparison to an agent which chooses actions uniformly at random.

   Make sure to name the files and Python dictionaries exactly as specified above and include your figure in your report pdf. Example: when $\epsilon = 0.1, \alpha = 0.6$, and $\gamma = 0.9$:

   ```
   python taxi_sarsa.py 0.1 0.6 0.9
   ```

b. Write Python code (from scratch) to implement the Q-learning algorithm in file `taxi_qlearning.py` and run your code on the Taxi environment to compute the optimal action-value function q^* and policy π^*. Your code must output three files:

   1. `qlearning_q_vals.pickle` which contains the dictionary `qlearning_q_vals` where the keys are the states (encoded exactly as given by the environment) and the values are dictionaries where the keys are the actions and the values are the learned q values $q(s, a)$
   2. `qlearning_policy.pickle` which contains a dictionary `qlearning_policy` where the keys are the states (encoded exactly as given by the environment) and the values are the policy actions (encoded as integers 0, 1, . . . , 5 exactly as the environment expects)
   3. a figure `qlearning_total_reward.png` which shows the total reward per episode (x-axis are the episodes and y-axis is the total reward per episode) in comparison to an agent which chooses actions uniformly at random. Make sure to name the files and Python dictionaries exactly as specified above and include your figure in your report pdf.

Problem 2. (30 points)[Programming] In this problem, you will implement a few multiagent learning algorithms to compete on a few repeated normal form games. We will consider the games:

	Testify	Refuse
Testify	1.0, 1.0	5.0, 0.0
Refuse	0.0, 5.0	3.0, 3.0

Prisoner's Dilemma

	Swerve	Straight
Swerve	3.0, 3.0	1.5, 3.5
Straight	3.5, 1.5	1.0, 1.0

Chicken

	Action	Comedy
Action	3.0, 2.0	0.0, 0.0
Comedy	0.0, 0.0	2.0, 3.0

Movie Selection

Prisoner's Dilemma (Table 1), Chicken (Table 2), and Movie Coordination (Table 3). In the tables, the row player's reward is given first followed by the column player's reward. Write Python code to implement (from scratch) the following agents/strategies: Tit-for-tat, Fictitious Play, Bully, and Godfather in file `multiagent_learning.py`. Apply your code for each of the algorithms to each of the 3 games above by having each agent compete against every other agent and also self-play. So, for example, agent Bully will compete against Tit-for-tat, Fictitious Play, Godfather, and itself (in self-play) in each of the 3 games above. Each competition must run for exactly T = 100 rounds (but the agents do not know this parameter). Your code should output the average reward received for each agent in each game. So, for each game, you should output a table where the rows and columns are the different agents and the elements are a tuple α, β where α is the average reward of the row agent and β is the average reward of the column agent. See the following example:

	Tit-for-tat	Fictitious Play	Bully	Godfather
Tit-for-tat	α, β			
Fictitious Play	-			
Bully	-	-		
Godfather	-	-	-	

Your Python code should output 3 tables (like above), one for each of the 3 games; no input is required (you can hardcode the number of rounds T = 100). Make sure you include those tables in your report pdf.