- NAME: RISHABH PATIL
- SAP: 60009200056
- BATCH: K2

```python
import numpy as np
import gym
import numpy as np
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```python
env = gym.make("FrozenLake-v1")
n_observations = env.observation_space.n
n_actions = env.action_space.n
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
/usr/local/lib/python3.9/dist-packages/gym/core.py:317: DeprecationWarning: WARN: Initializing wrapper in old step API which returns one bool instead of two. It is recommended to set `new_step_api=True` to use new step API. This will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.9/dist-packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning: WARN: Initializing environment in old step API which returns one bool instead of two. It is recommended to set `new_step_api=True` to use new step API. This will be the default behaviour in future.
  deprecation(
```

```python
#Initialize the Q-table to 0
Q_table = np.zeros((n_observations,n_actions))
print(Q_table)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
```

 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

```python
#number of episode we will run
n_episodes = 10000

#maximum of iteration per episode
max_iter_episode = 100

#initialize the exploration probability to 1
exploration_proba = 1

#exploartion decreasing decay for exponential decreasing
exploration_decreasing_decay = 0.001

# minimum of exploration proba
min_exploration_proba = 0.01

#discounted factor
gamma = 0.99

#learning rate
lr = 0.1

total_rewards_episode = list()

rewards_per_episode=[]
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to
`transformed_cell` argument and any exception that happen during
thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```python
#we iterate over episodes
for e in range(n_episodes):
    #we initialize the first state of the episode
    current_state = env.reset()
    done = False

    #sum the rewards that the agent gets from the environment
    total_episode_reward = 0

    for i in range(max_iter_episode):
        # we sample a float from a uniform distribution over 0 and 1
```

```
        # if the sampled flaot is less than the exploration proba
        #     the agent selects arandom action
        # else
        #     he exploits his knowledge using the bellman equation

        if np.random.uniform(0,1) < exploration_proba:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q_table[current_state,:])

        # The environment runs the chosen action and returns
        # the next state, a reward and true if the epiosed is ended.
        next_state, reward, done, _ = env.step(action)

        # We update our Q-table using the Q-learning iteration
        Q_table[current_state, action] = (1-lr) *
Q_table[current_state, action] +lr*(reward +
gamma*max(Q_table[next_state,:]))
        total_episode_reward = total_episode_reward + reward
        # If the episode is finished, we leave the for loop
        if done:
            break
        current_state = next_state
    #We update the exploration proba using exponential decay formula
    exploration_proba = max(min_exploration_proba, np.exp(-
exploration_decreasing_decay*e))
    rewards_per_episode.append(total_episode_reward)

print("Mean reward per thousand episodes")
for i in range(10):
    print((i+1)*1000,": mean espiode reward:",
np.mean(rewards_per_episode[1000*i:1000*(i+1)]))
```

```
Mean reward per thousand episodes
1000 : mean espiode reward: 0.046
2000 : mean espiode reward: 0.226
3000 : mean espiode reward: 0.424
4000 : mean espiode reward: 0.593
5000 : mean espiode reward: 0.653
6000 : mean espiode reward: 0.682
7000 : mean espiode reward: 0.679
8000 : mean espiode reward: 0.671
9000 : mean espiode reward: 0.669
10000 : mean espiode reward: 0.694
```