In [1]:
```
import numpy as np
import gym
import random
```

In [2]:
```
# initiate an env for RL

env = gym.make("FrozenLake-v1",map_name ='4x4',is_slippery = False)
print("Observation space:", env.observation_space )
```

```
Observation space: Discrete(16)
```

In [3]:
```
print("Action space shape", env.action_space.n)
```

```
Action space shape 4
```

In [4]:
```
state_space = env.observation_space.n
action_space = env.action_space.n
```

In [5]:
```
# Create and initiate Q table

def initialize_q_table(state_space, action_space):
    qtable = np.zeros((state_space, action_space))
    return qtable

Qtable_frozenlake = initialize_q_table(state_space,action_space)
Qtable_frozenlake
```

Out[5]:
```
array([[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.]])
```

In [13]:
```
# epsilon greedy policy

def epsilon_greedy_policy(qtable,state,epsilon):
    random_int = random.uniform(0,1)
    if random_int>epsilon:
        action = np.argmax(qtable[state])
    else:
        action = env.action_space.sample()
    return action
```

In [7]:

```python
# greedy policy

def greedy_policy(qtable,state):
    action = np.argmax(qtable[state])
    return action
```

In [8]:

```python
# model hyperparameters

n_training_episodes = 10000
learning_rate = 0.7
n_eval_episodes = 100

env_id = "FrozenLake-v1"
max_steps = 99
gamma = 0.95

max_epsilon = 1.0
min_epsilon  = 0.05
decay_rate = 0.0005
```

In [10]:

```python
from tqdm.notebook import trange
```

In [15]:

```
def train(n_training_episodes,min_epsilon,max_epsilon,decay_rate,env,max_steps,qtable):
    for episode in trange(n_training_episodes):
        epsilon = min_epsilon + (max_epsilon-min_epsilon)*np.exp(-decay_rate*episode)
        state = env.reset()
        step = 0
        done = False

        for step in range(max_steps):
            action = epsilon_greedy_policy(qtable,state,epsilon)
            new_state,reward,done,info = env.step(action)
            qtable[state][action] = qtable[state][action] + learning_rate * (reward + gamma * np.max
                                                                             qtable[state][action])
            # if done finish episode
            if done:
                break

            state = new_state
    return qtable
```

In [16]:
```
frozenlake = train(n_training_episodes, min_epsilon, max_epsilon, decay_rate, env, max_steps, Qtable
frozenlake
```

```
100%                                                  10000/10000 [00:05<00:00, 2421.83it/s]
```

Out[16]:
```
array([[0.73509189, 0.77378094, 0.77378094, 0.73509189],
       [0.73509189, 0.        , 0.81450625, 0.77378094],
       [0.77378094, 0.857375  , 0.77378094, 0.81450625],
       [0.81450625, 0.        , 0.77378094, 0.77378094],
       [0.77378094, 0.81450625, 0.        , 0.73509189],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.9025    , 0.        , 0.81450625],
       [0.        , 0.        , 0.        , 0.        ],
       [0.81450625, 0.        , 0.857375  , 0.77378094],
       [0.81450625, 0.9025    , 0.9025    , 0.        ],
       [0.857375  , 0.95      , 0.        , 0.857375  ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.9025    , 0.95      , 0.857375  ],
       [0.9025    , 0.95      , 1.        , 0.9025    ],
       [0.        , 0.        , 0.        , 0.        ]])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: