NNFS LAB 1: Intelligent Agents

Reinforcement Learning - taxi problem

Group 1:

Parth Deshpande 191060022

Ishan Deshpande 191060021

Dev Sharma 191060023

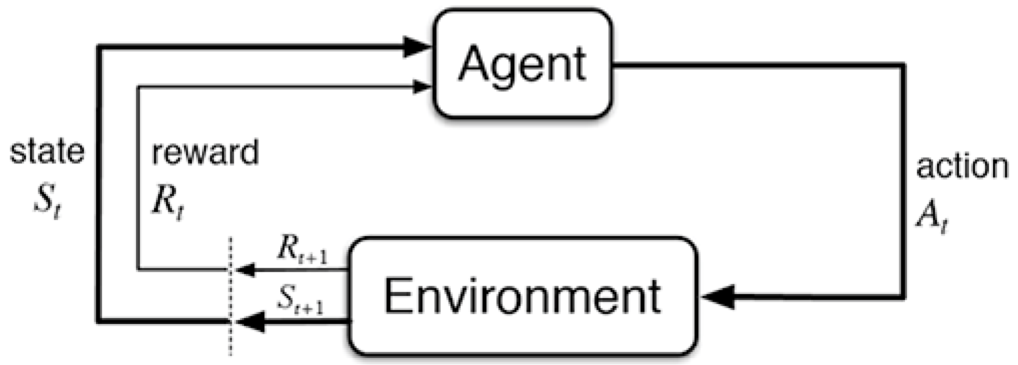
Kedar Daithankar 191060019

Reinforcement Learning:

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. It involves an agent interacting with its surrounding environment to determine what is the best action to take. The environment could be uncertain, complex, and the agent's behaviour can also be probabilistic, not deterministic.

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Taxi is one of many environments available on OpenAI Gym. These environments are used to develop and benchmark reinforcement learning algorithms.



In a RL problem, the agent would want to maximize its total rewards. It constantly interacts with the environment and explore to gather information about what reward it gets from executing an action as specific states, then search for the optimal action to take at each state.

Q Learning is a type of Value-based learning algorithms. The agent's objective is to optimize a "Value function" suited to the problem it faces. In a Q learning process we have a **Q-table** that stores the Q value for each state and each possible action, the agent explores the environment and make update to the Q values iteratively.

Learning Process:

Step 1: Initialize all Q values in the Q- table to 0, the agent has no knowledge about the environment it is in.

Step 2: Explore the space: The agent keeps exploring the environment by executing actions at the states it is in.

Exploration vs Exploitation:

We can let our agent explore to update our Q-table using the Q-learning algorithm. As our agent learns more about the environment, we can let it use this knowledge to take more optimal actions and converge faster-known as exploitation. During exploitation, our agent will look at its Q-table and select the action with the highest Q-value (instead of a random action). Over time, our agent will need to explore less, and start exploiting what it knows instead.

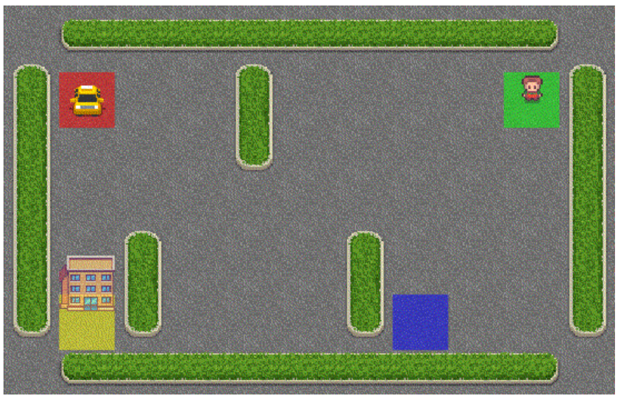
Step 3: observe the reward - When exploring, the agent would observe what reward it gets from executing a particular action (at) in state (st) to go to next state (st+1).

Step 4: After observing the reward, the agent then updates the value function for the particular state and action pair using the following formula, this returns an updated Q-table.

$$\overset{\text{updated value}}{Q(st,at)} = \underbrace{Q(st,at)}_{\text{Original value}} + \overset{\text{learning rate}}{\alpha} {}^{*} \left(\underbrace{Rt(s,a)}_{\text{reward we get for the current state and action}} + \overset{\text{discount rate}}{\gamma} {}^{*} \left(\underbrace{\max_{a} Q(st+1,a)}_{\text{maximum future value in state s+1 - if we take the best action a}}\right) - \underbrace{Q(st,at)}_{\text{Original value}}\right)$$

TAXI PROBLEM:

There are four designated locations in the grid world indicated by **Red**, **Green**, **Yellow**, and **Blue**. When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the four specified locations), and then drops off the passenger. Once the passenger is dropped off, the episode ends.



Actions

There are **6 discrete deterministic actions**:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

There are **500 discrete states** since there are **25 taxi positions**, 5 possible locations of the passenger

Passenger locations:

- 0: Red
- 1: Green
- 2: Yellow
- 3: Blue
- 4: in taxi

Destinations:

- 0: Red
- 1: Green
- 2: Yellow
- 3: Blue

Rewards

-1 per step unless other reward is triggered.

+20 delivering passenger.

-10 executing "pickup" and "drop-off" actions illegally.

Q-tables:

For storing the points/Maximum expected future rewards we use q-tables. Q-table consists of number of **rows** same as the number of states of the agent in that environment & number of **columns** are the total number of actions it can take.

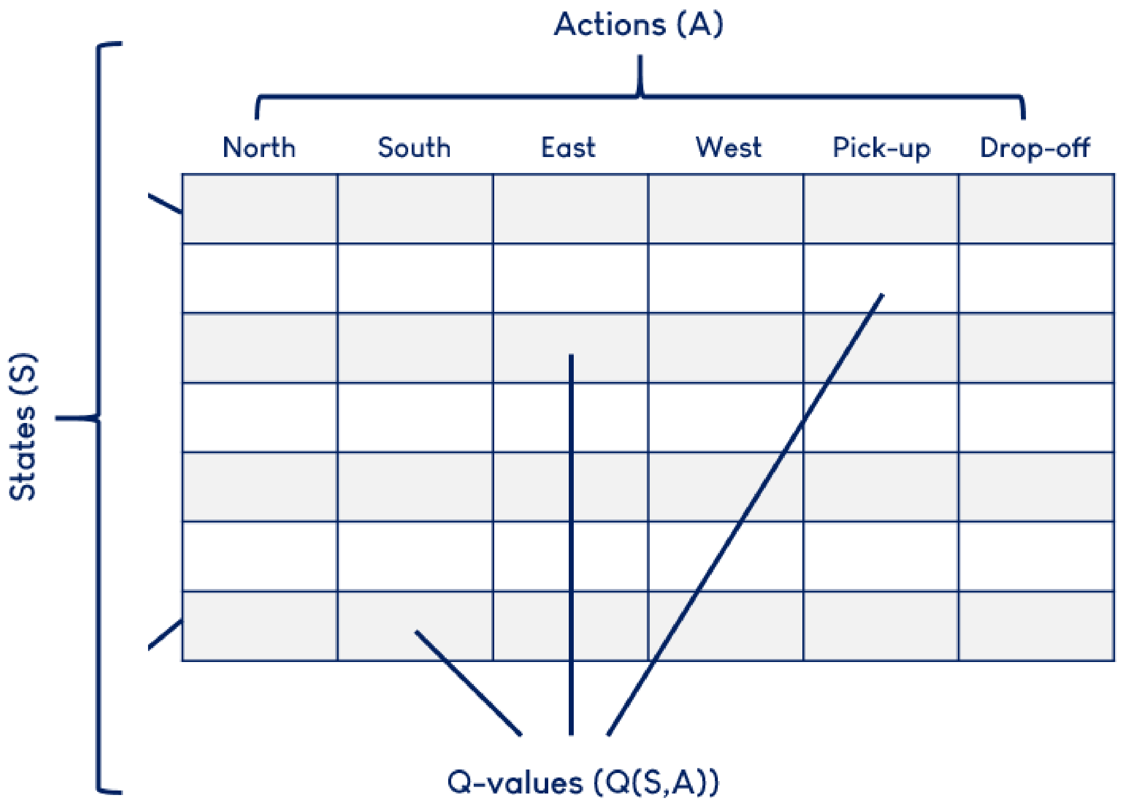
In this example, Agent can have 500 states and it can perform 6 actions as explained above.

So, the dimensions of q-table are **(500 x 6)**

Each row corresponds to a unique state in the 'Taxi' environment

Each column corresponds to an action our agent can take

Each cell corresponds to the Q-value for that state-action pair - a higher Q-value means a higher maximum reward our agent can expect to get if it takes that action in that state.



The Q-learning algorithm will help our agent update the current Q-value (Q(St,At)) with its observations after taking an action. I.e. increase Q if it encountered a positive reward, or decrease Q if it encountered a negative one.

Why discount factor?

The discount factor (a number between 0-1) is a clever way to scale down the rewards more and more after each step so that, the total sum remains bounded.

Implementation:

https://github.com/heisenberg-88/nnfs_lab1_Reinforcement-Learning---taxi-problem

There were some changes in the recent versions Taxi-V3 api, so we had to change the code accordingly.

In the new version, to make the environment work, we have added render_mode as an additional argument according to the documentation.

```
streets = gym.make("Taxi-v3",render_mode="human").env
```

also,
we have to call env.reset() before any env.render()

In every episode, we have to call env.reset().In new taxi-v3,
this function returns **current state number** & other info, probability, etc

```
state, info = streets.reset(
```

the state value ranges from 1 to 500
we can access q-table data using this**. q_table[state].**
This returns a numpy array consisting of 6 columns corresponding to 6 actions.

The process for training our agent will look like:

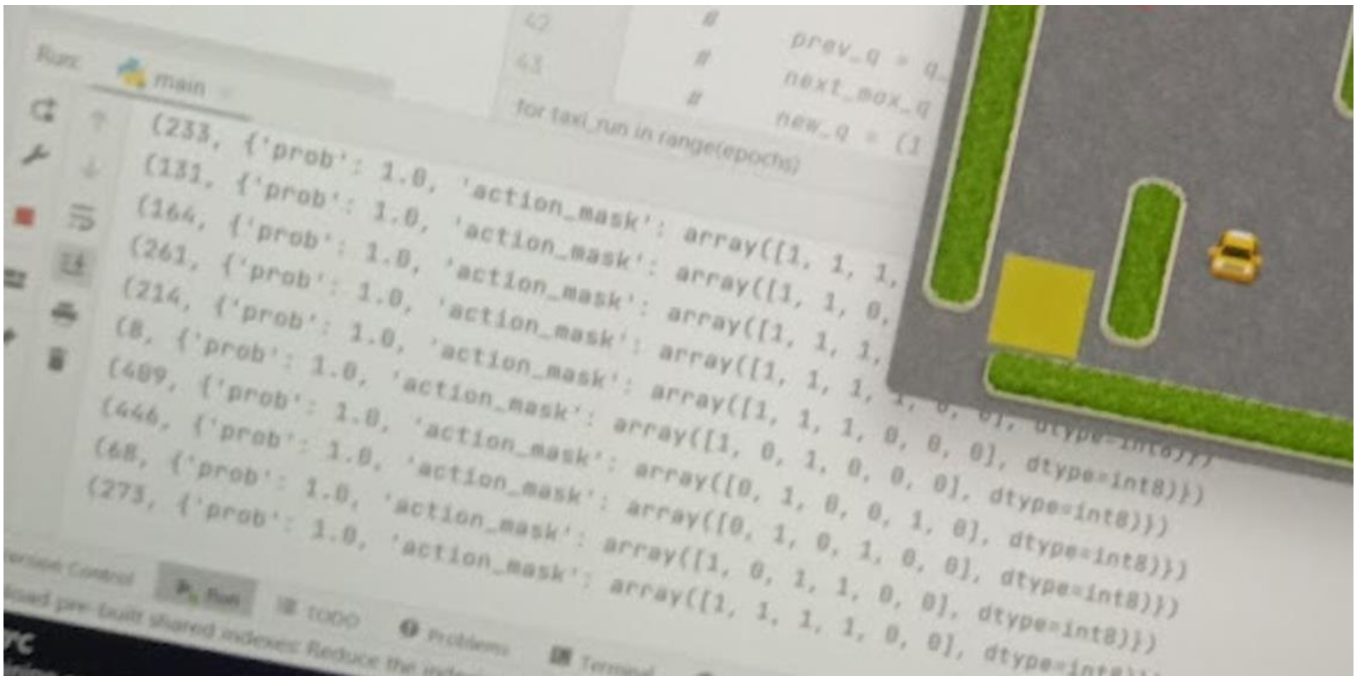
- Initialising our Q-table with 0's for all Q-values.
- Let our agent play Taxi over a large number of games.
- Continuously update the Q-table using the Q-learning algorithm and an exploration-exploitation strategy.

Here's the full implementation:

Printway function is used for printing the steps.

```python
def printway(index):
    value = ""
    if index==0:
        value = "down"
    elif index==1:
        value ="up"
    elif index==2:
        value ="right"
    elif index==3:
        value ="left"
    elif index==4:
        value ="pickup passenger"
    elif index==5:
        value ="drop off passenger"
    return value
```

Showing the state number and the corresponding action array.



```
#            prev_q = q
#            next_max_q
#            new_q = (1
for taxi_run in range(epochs)
```

```
(233, {'prob': 1.0, 'action_mask': array([1, 1, 1,
(131, {'prob': 1.0, 'action_mask': array([1, 1, 0,
(164, {'prob': 1.0, 'action_mask': array([1, 1, 1,
(261, {'prob': 1.0, 'action_mask': array([1, 1, 1,
(214, {'prob': 1.0, 'action_mask': array([1, 1, 1, 1, 0, 0], dtype=int8)})
(8, {'prob': 1.0, 'action_mask': array([1, 0, 1, 0, 0, 0], dtype=int8)})
(489, {'prob': 1.0, 'action_mask': array([0, 1, 0, 0, 1, 0], dtype=int8)})
(446, {'prob': 1.0, 'action_mask': array([0, 1, 0, 1, 0, 0], dtype=int8)})
(68, {'prob': 1.0, 'action_mask': array([1, 0, 1, 1, 0, 0], dtype=int8)})
(273, {'prob': 1.0, 'action_mask': array([1, 1, 1, 1, 0, 0], dtype=int8)})
```

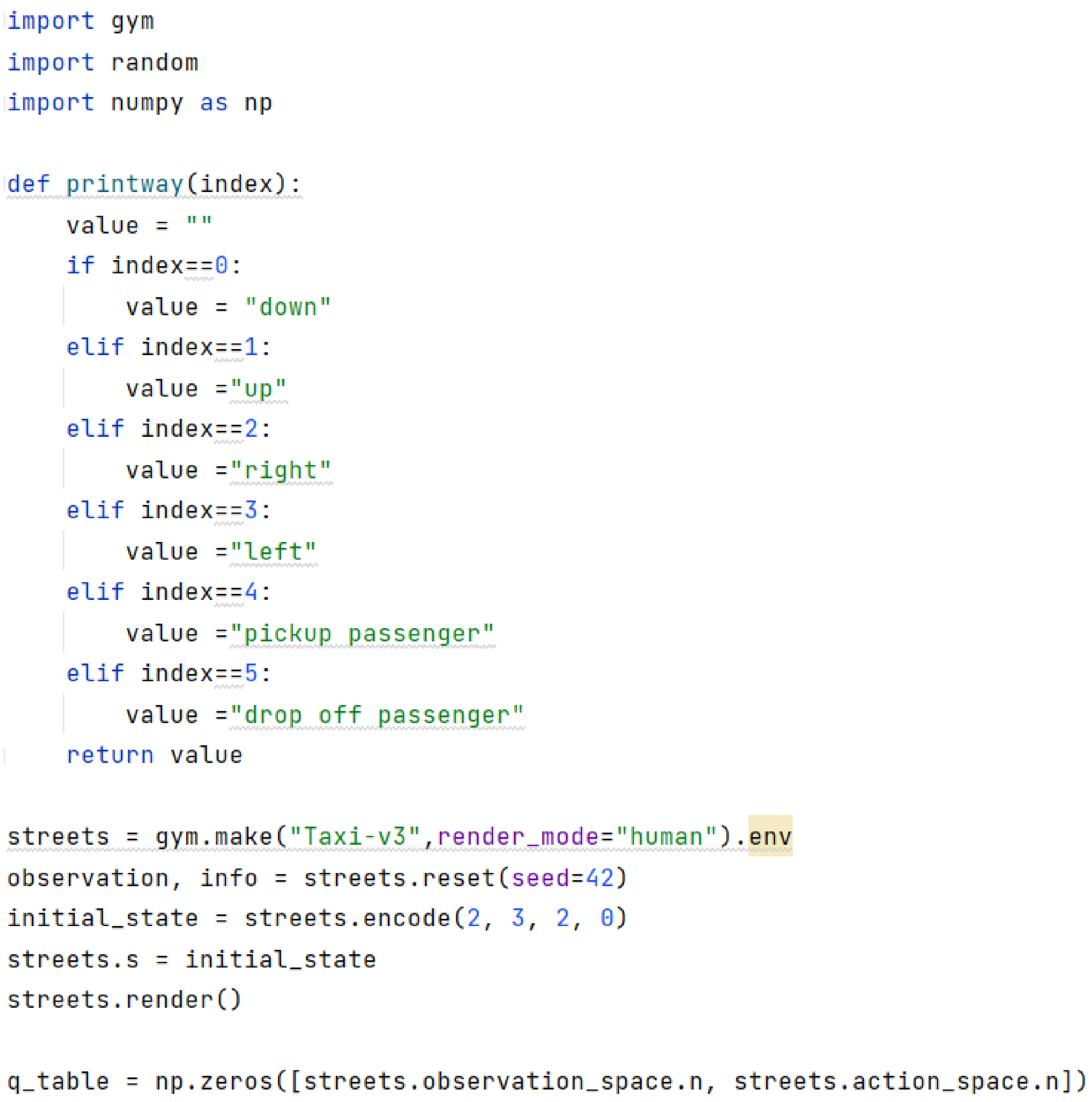
We have created two functions for 2 different scenarios:

1) q_learning_new
2) q_learning_precalc

q_learning_new is used to compute the q_table when the agent does not have any prior exploration.

q_learning_precalc is used when we have q_table updated for some number of epochs. This function can be used when we want to start the q learning from a existing q_table.

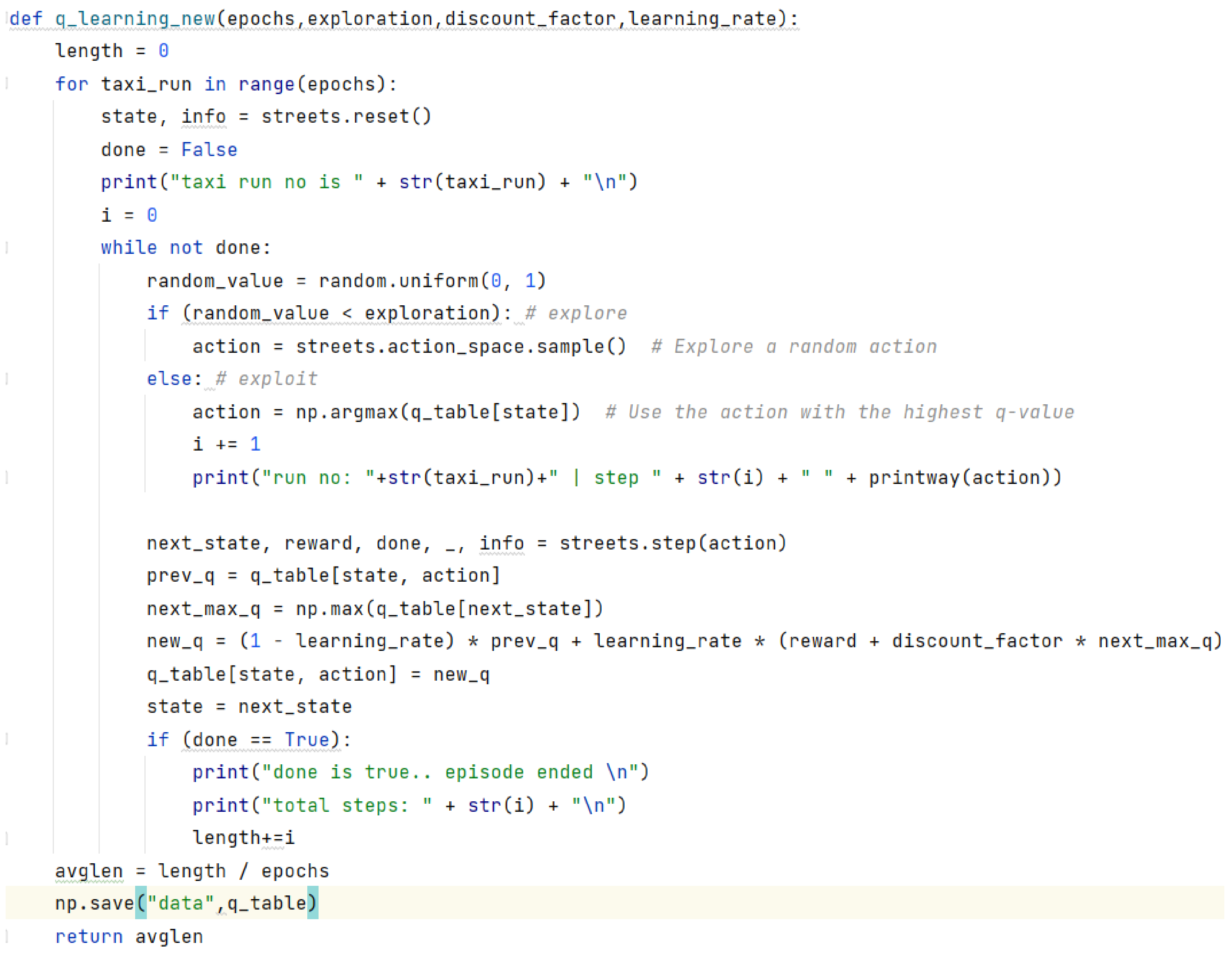
q_learning_new.py



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

def printway(index):
    value = ""
    if index==0:
        value = "down"
    elif index==1:
        value ="up"
    elif index==2:
        value ="right"
    elif index==3:
        value ="left"
    elif index==4:
        value ="pickup passenger"
    elif index==5:
        value ="drop off passenger"
    return value

streets = gym.make("Taxi-v3",render_mode="human").env
observation, info = streets.reset(seed=42)
initial_state = streets.encode(2, 3, 2, 0)
streets.s = initial_state
streets.render()

q_table = np.zeros([streets.observation_space.n, streets.action_space.n])
```

here q_table with zeros is created for initialization.

Streets taxiV3 environment is created from gym.make().



```python
def q_learning_new(epochs,exploration,discount_factor,learning_rate):
    length = 0
    for taxi_run in range(epochs):
        state, info = streets.reset()
        done = False
        print("taxi run no is " + str(taxi_run) + "\n")
        i = 0
        while not done:
            random_value = random.uniform(0, 1)
            if (random_value < exploration): # explore
                action = streets.action_space.sample()  # Explore a random action
            else: # exploit
                action = np.argmax(q_table[state])  # Use the action with the highest q-value
                i += 1
                print("run no: "+str(taxi_run)+" | step " + str(i) + " " + printway(action))

            next_state, reward, done, _, info = streets.step(action)
            prev_q = q_table[state, action]
            next_max_q = np.max(q_table[next_state])
            new_q = (1 - learning_rate) * prev_q + learning_rate * (reward + discount_factor * next_max_q)
            q_table[state, action] = new_q
            state = next_state
            if (done == True):
                print("done is true.. episode ended \n")
                print("total steps: " + str(i) + "\n")
                length+=i
    avglen = length / epochs
    np.save("data",q_table)
    return avglen
```

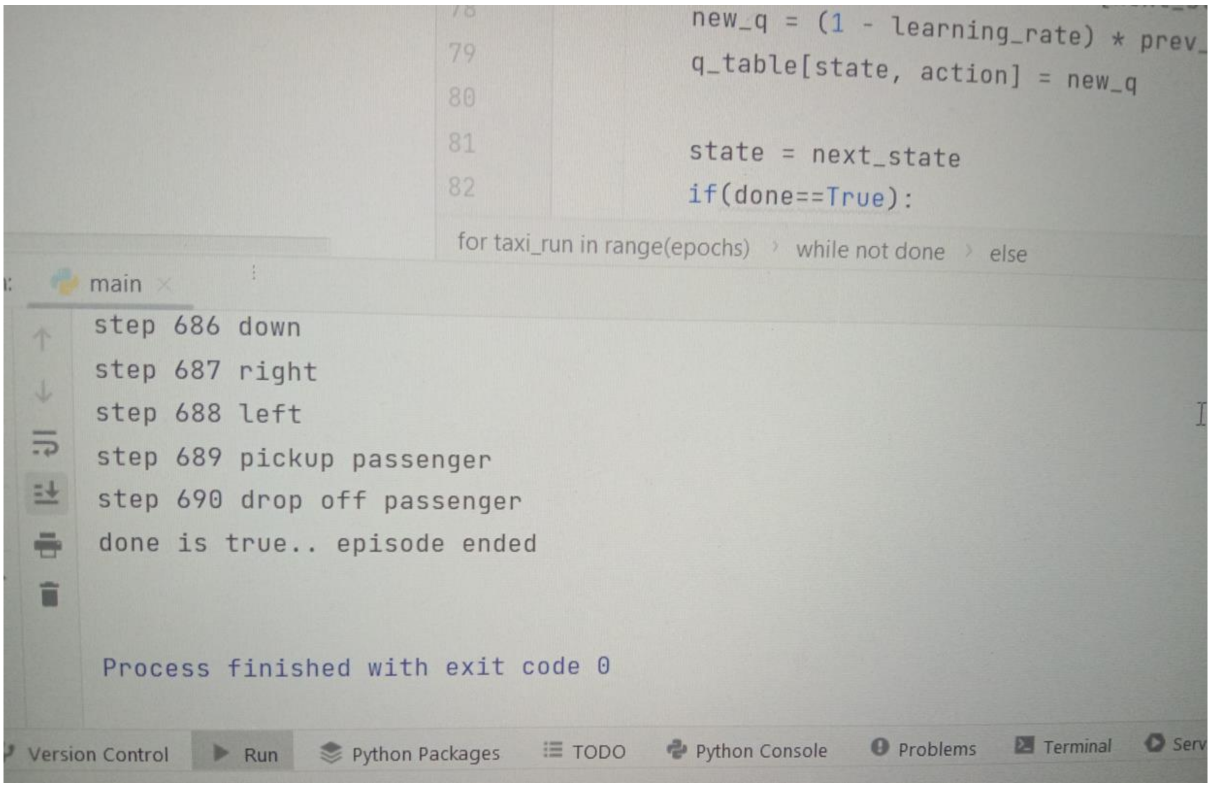
This function returns the average length avglen of the steps taken during each episode for all epochs. Adds number of steps ‘i’ to the lengths and then divides by epochs. `length+=i ,avglen = length/epochs`
This shows the current epoch and the steps/actions that agent is taking.

```
print("run no: "+str(taxi_run)+" | step " + str(i) + " " +printway(action))
```

in old taxiv3, step() returns 4 values with the done boolean. But, in new taxiv3 api, env.step() returns 5 values: Observation – next_state, Reward, terminated – done bool replaced, truncated, info.
To get proper return values, changes are made accordingly.

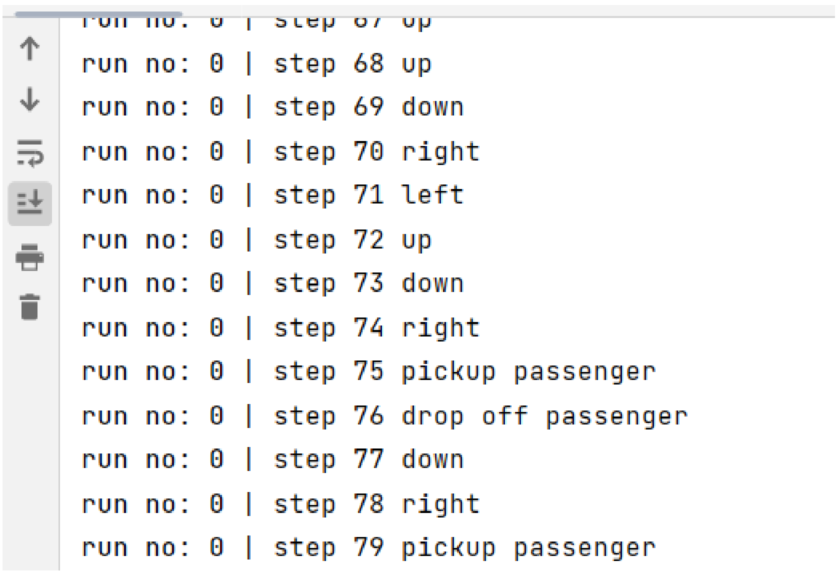
```
next_state, reward, done, _, info = streets.step(action)
```

Steps showing till each episode ends



```
            new_q = (1 - learning_rate) * prev_
            q_table[state, action] = new_q

            state = next_state
            if(done==True):
```

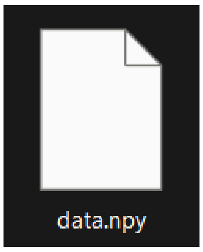
```
step 686 down
step 687 right
step 688 left
step 689 pickup passenger
step 690 drop off passenger
done is true.. episode ended


Process finished with exit code 0
```



```
run no: 0 | step 68 up
run no: 0 | step 69 down
run no: 0 | step 70 right
run no: 0 | step 71 left
run no: 0 | step 72 up
run no: 0 | step 73 down
run no: 0 | step 74 right
run no: 0 | step 75 pickup passenger
run no: 0 | step 76 drop off passenger
run no: 0 | step 77 down
run no: 0 | step 78 right
run no: 0 | step 79 pickup passenger
```

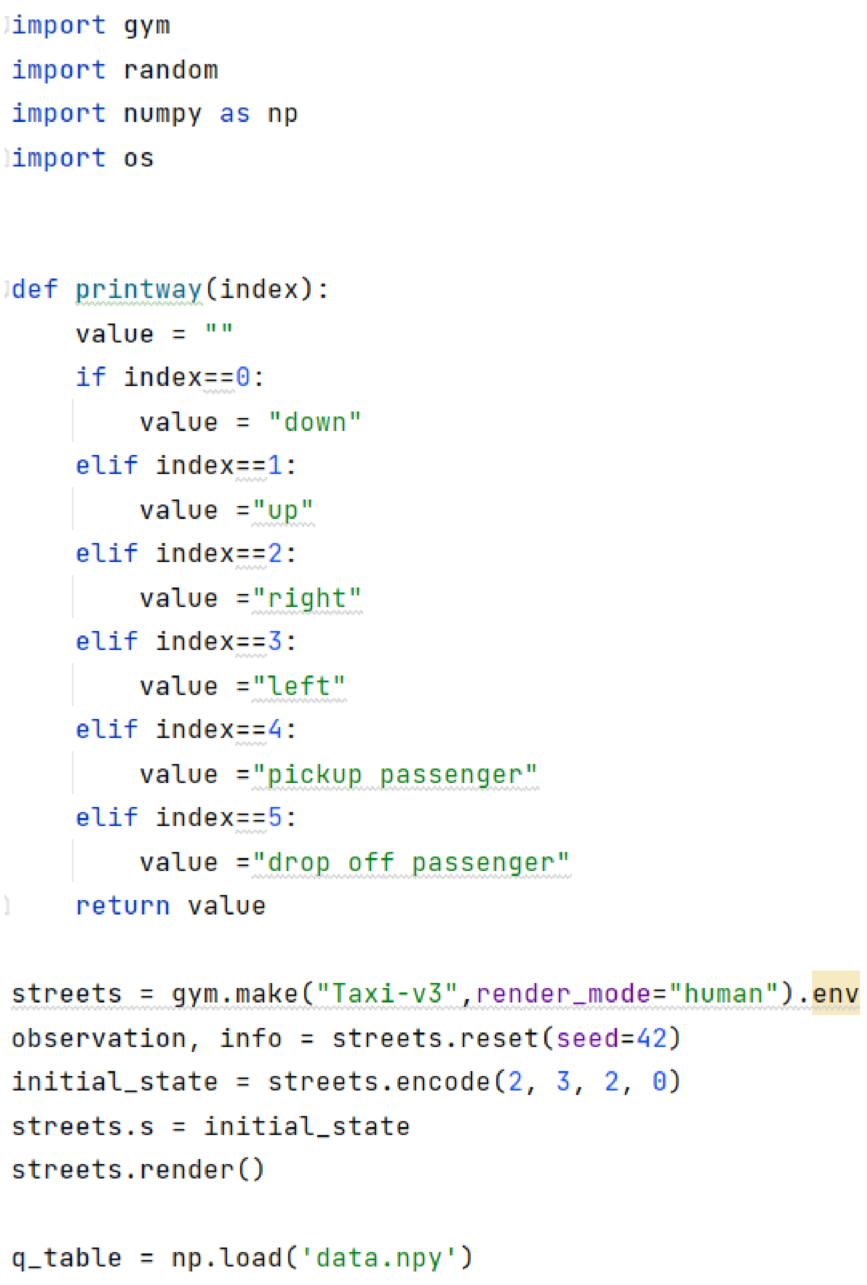
After all epochs/taxi runs, the q_table is stored in data.npy format.

```
np.save("data",q_table)
```



q_learning_precalc.py

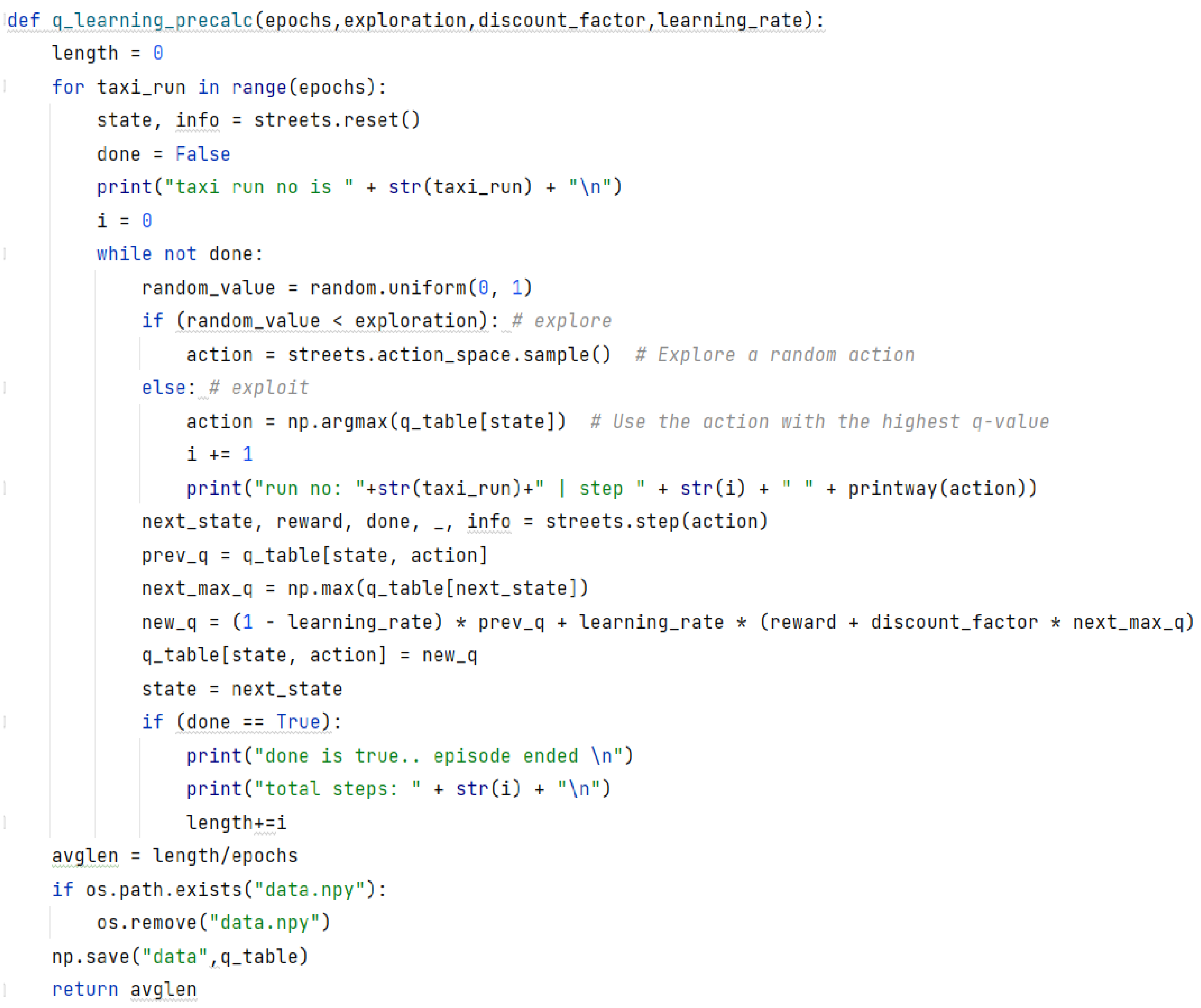
This is same as previous one but the change is that , we can use this if we want to use previously generated q_table and continue learning from that instance.



```
import gym
import random
import numpy as np
import os


def printway(index):
    value = ""
    if index==0:
        value = "down"
    elif index==1:
        value ="up"
    elif index==2:
        value ="right"
    elif index==3:
        value ="left"
    elif index==4:
        value ="pickup passenger"
    elif index==5:
        value ="drop off passenger"
    return value

streets = gym.make("Taxi-v3",render_mode="human").env
observation, info = streets.reset(seed=42)
initial_state = streets.encode(2, 3, 2, 0)
streets.s = initial_state
streets.render()

q_table = np.load('data.npy')
```

Here we are loading the precomputed q_table using

```
np.load('data.npy')
```



```python
def q_learning_precalc(epochs,exploration,discount_factor,learning_rate):
    length = 0
    for taxi_run in range(epochs):
        state, info = streets.reset()
        done = False
        print("taxi run no is " + str(taxi_run) + "\n")
        i = 0
        while not done:
            random_value = random.uniform(0, 1)
            if (random_value < exploration): # explore
                action = streets.action_space.sample()  # Explore a random action
            else: # exploit
                action = np.argmax(q_table[state])  # Use the action with the highest q-value
                i += 1
                print("run no: "+str(taxi_run)+" | step " + str(i) + " " + printway(action))
            next_state, reward, done, _, info = streets.step(action)
            prev_q = q_table[state, action]
            next_max_q = np.max(q_table[next_state])
            new_q = (1 - learning_rate) * prev_q + learning_rate * (reward + discount_factor * next_max_q)
            q_table[state, action] = new_q
            state = next_state
            if (done == True):
                print("done is true.. episode ended \n")
                print("total steps: " + str(i) + "\n")
                length+=i
    avglen = length/epochs
    if os.path.exists("data.npy"):
        os.remove("data.npy")
    np.save("data",q_table)
    return avglen
```

avglen is returned by calculating length/epochs. Length is incremented at every step.

At the end of an episode total steps are added to length

```
length+=i
```

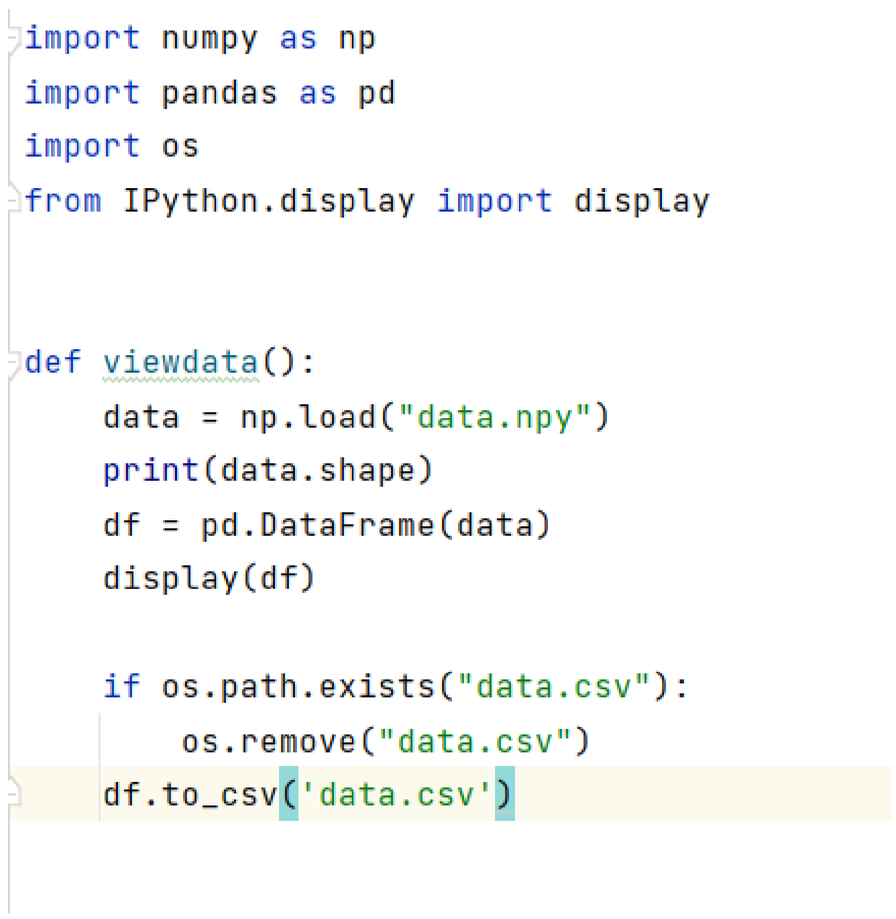
and then average is calculated

```
avglen = length/epochs
```

here, the old q_table npy file is deleted and new npy data of q_table is created.

```python
if os.path.exists("data.npy"):
    os.remove("data.npy")
np.save("data",q_table)
```

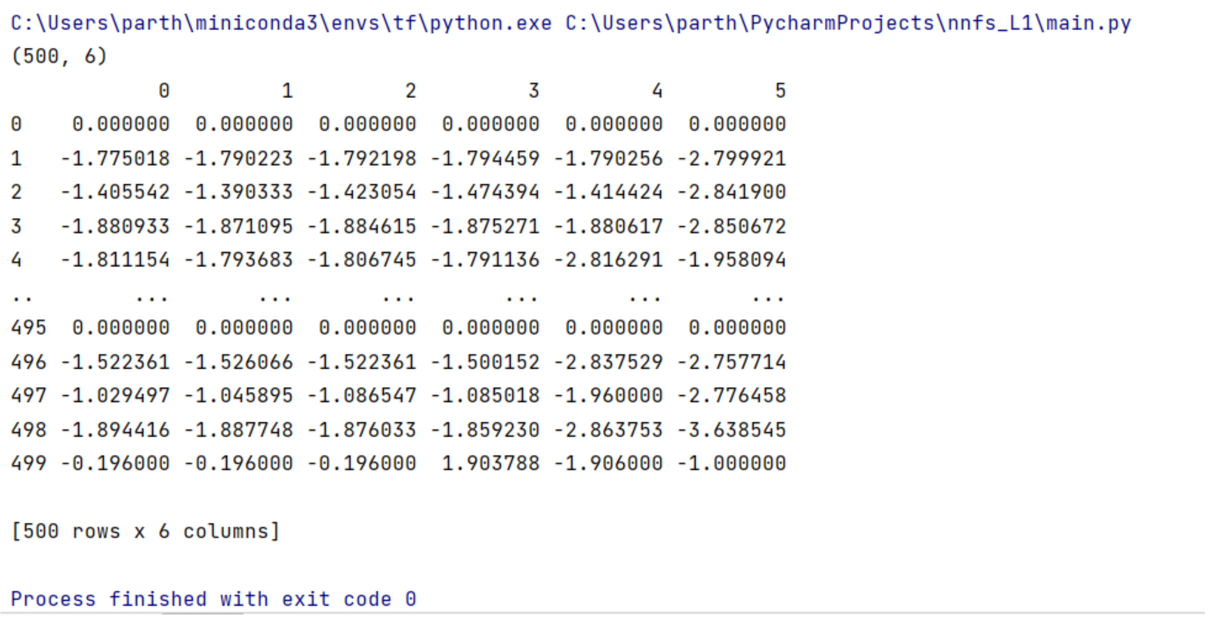
viewdata.py



```
import numpy as np
import pandas as pd
import os
from IPython.display import display


def viewdata():
    data = np.load("data.npy")
    print(data.shape)
    df = pd.DataFrame(data)
    display(df)

    if os.path.exists("data.csv"):
        os.remove("data.csv")
    df.to_csv('data.csv')
```

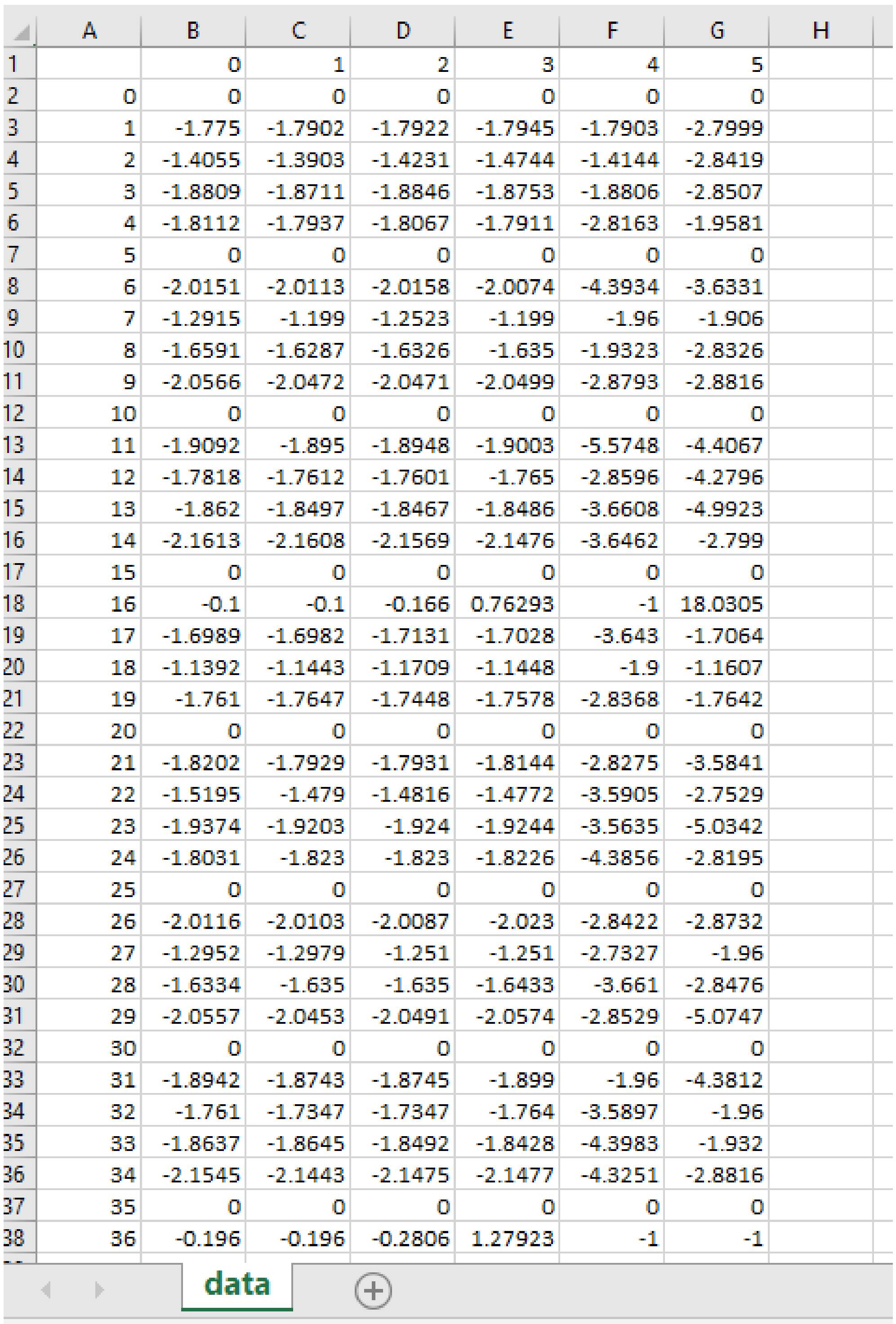
Viewdata() function is used to create and show the q_table dataframe. Q_table can also be exported to csv format using this function for further use.

q_table



```
C:\Users\parth\miniconda3\envs\tf\python.exe C:\Users\parth\PycharmProjects\nnfs_L1\main.py
(500, 6)
            0         1         2         3         4         5
0    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1   -1.775018 -1.790223 -1.792198 -1.794459 -1.790256 -2.799921
2   -1.405542 -1.390333 -1.423054 -1.474394 -1.414424 -2.841900
3   -1.880933 -1.871095 -1.884615 -1.875271 -1.880617 -2.850672
4   -1.811154 -1.793683 -1.806745 -1.791136 -2.816291 -1.958094
..        ...       ...       ...       ...       ...       ...
495  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
496 -1.522361 -1.526066 -1.522361 -1.500152 -2.837529 -2.757714
497 -1.029497 -1.045895 -1.086547 -1.085018 -1.960000 -2.776458
498 -1.894416 -1.887748 -1.876033 -1.859230 -2.863753 -3.638545
499 -0.196000 -0.196000 -0.196000  1.903788 -1.906000 -1.000000

[500 rows x 6 columns]

Process finished with exit code 0
```

csv q_table data:

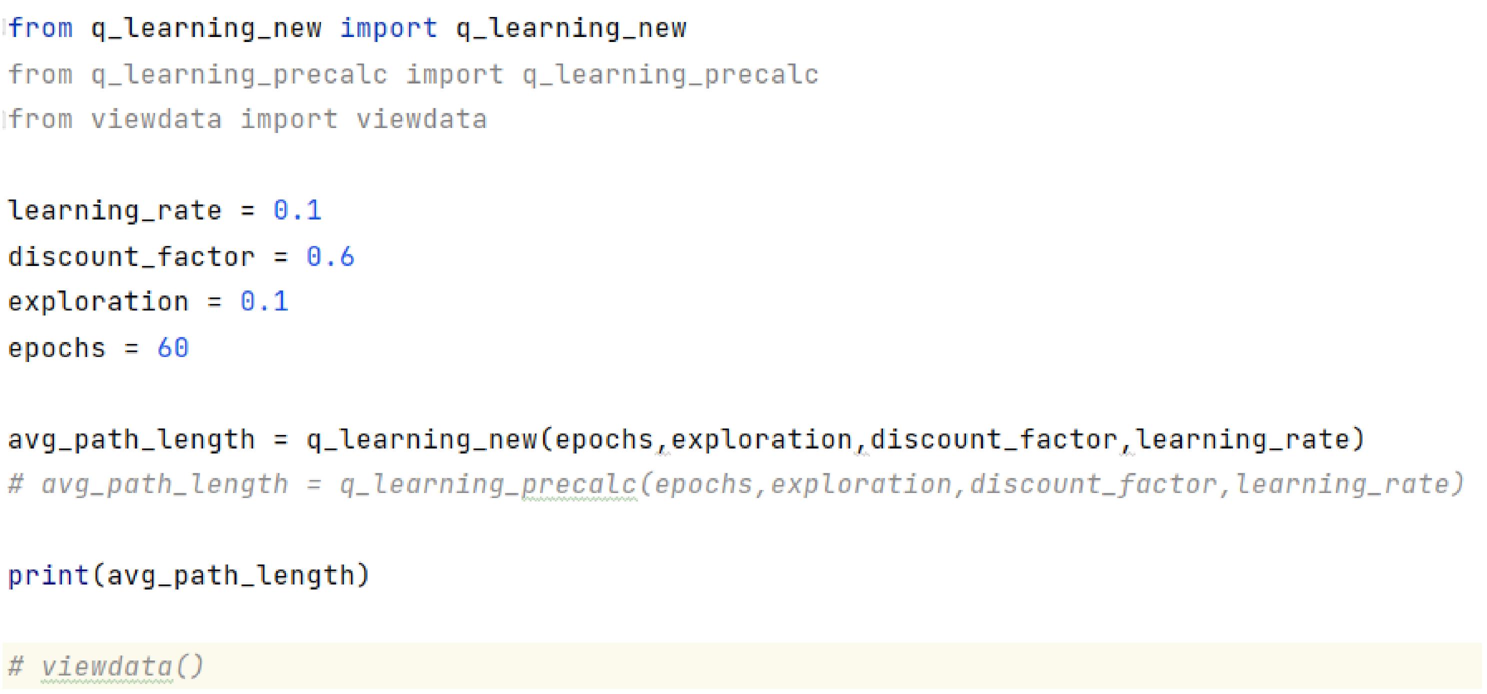


	A	B	C	D	E	F	G	H
1		0	1	2	3	4	5	
2	0	0	0	0	0	0	0	
3	1	-1.775	-1.7902	-1.7922	-1.7945	-1.7903	-2.7999	
4	2	-1.4055	-1.3903	-1.4231	-1.4744	-1.4144	-2.8419	
5	3	-1.8809	-1.8711	-1.8846	-1.8753	-1.8806	-2.8507	
6	4	-1.8112	-1.7937	-1.8067	-1.7911	-2.8163	-1.9581	
7	5	0	0	0	0	0	0	
8	6	-2.0151	-2.0113	-2.0158	-2.0074	-4.3934	-3.6331	
9	7	-1.2915	-1.199	-1.2523	-1.199	-1.96	-1.906	
10	8	-1.6591	-1.6287	-1.6326	-1.635	-1.9323	-2.8326	
11	9	-2.0566	-2.0472	-2.0471	-2.0499	-2.8793	-2.8816	
12	10	0	0	0	0	0	0	
13	11	-1.9092	-1.895	-1.8948	-1.9003	-5.5748	-4.4067	
14	12	-1.7818	-1.7612	-1.7601	-1.765	-2.8596	-4.2796	
15	13	-1.862	-1.8497	-1.8467	-1.8486	-3.6608	-4.9923	
16	14	-2.1613	-2.1608	-2.1569	-2.1476	-3.6462	-2.799	
17	15	0	0	0	0	0	0	
18	16	-0.1	-0.1	-0.166	0.76293	-1	18.0305	
19	17	-1.6989	-1.6982	-1.7131	-1.7028	-3.643	-1.7064	
20	18	-1.1392	-1.1443	-1.1709	-1.1448	-1.9	-1.1607	
21	19	-1.761	-1.7647	-1.7448	-1.7578	-2.8368	-1.7642	
22	20	0	0	0	0	0	0	
23	21	-1.8202	-1.7929	-1.7931	-1.8144	-2.8275	-3.5841	
24	22	-1.5195	-1.479	-1.4816	-1.4772	-3.5905	-2.7529	
25	23	-1.9374	-1.9203	-1.924	-1.9244	-3.5635	-5.0342	
26	24	-1.8031	-1.823	-1.823	-1.8226	-4.3856	-2.8195	
27	25	0	0	0	0	0	0	
28	26	-2.0116	-2.0103	-2.0087	-2.023	-2.8422	-2.8732	
29	27	-1.2952	-1.2979	-1.251	-1.251	-2.7327	-1.96	
30	28	-1.6334	-1.635	-1.635	-1.6433	-3.661	-2.8476	
31	29	-2.0557	-2.0453	-2.0491	-2.0574	-2.8529	-5.0747	
32	30	0	0	0	0	0	0	
33	31	-1.8942	-1.8743	-1.8745	-1.899	-1.96	-4.3812	
34	32	-1.761	-1.7347	-1.7347	-1.764	-3.5897	-1.96	
35	33	-1.8637	-1.8645	-1.8492	-1.8428	-4.3983	-1.932	
36	34	-2.1545	-2.1443	-2.1475	-2.1477	-4.3251	-2.8816	
37	35	0	0	0	0	0	0	
38	36	-0.196	-0.196	-0.2806	1.27923	-1	-1	

data

……… till 500 rows (500 states)

main.py



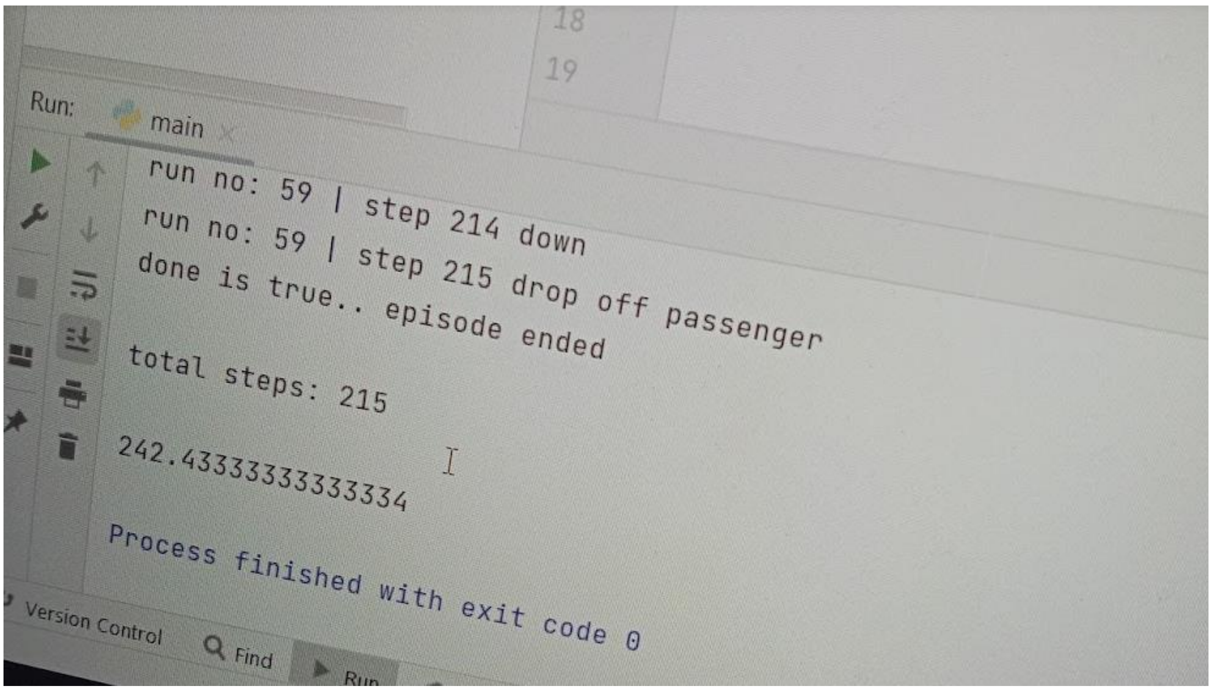
```
from q_learning_new import q_learning_new
from q_learning_precalc import q_learning_precalc
from viewdata import viewdata

learning_rate = 0.1
discount_factor = 0.6
exploration = 0.1
epochs = 60

avg_path_length = q_learning_new(epochs,exploration,discount_factor,learning_rate)
# avg_path_length = q_learning_precalc(epochs,exploration,discount_factor,learning_rate)

print(avg_path_length)

# viewdata()
```

here, we’ve imported the 2 functions from 2 files mentioned above (i.e. q_learning_new.py & q_learning_precalc.py)

and viewdata.py file is also imported to interact with q_table.

Avg length of steps returned from q_learning functions is stored in avg_path_length.

here avg length is 242 steps.



```
run no: 59 | step 214 down
run no: 59 | step 215 drop off passenger
done is true.. episode ended

total steps: 215

242.43333333333334

Process finished with exit code 0
```

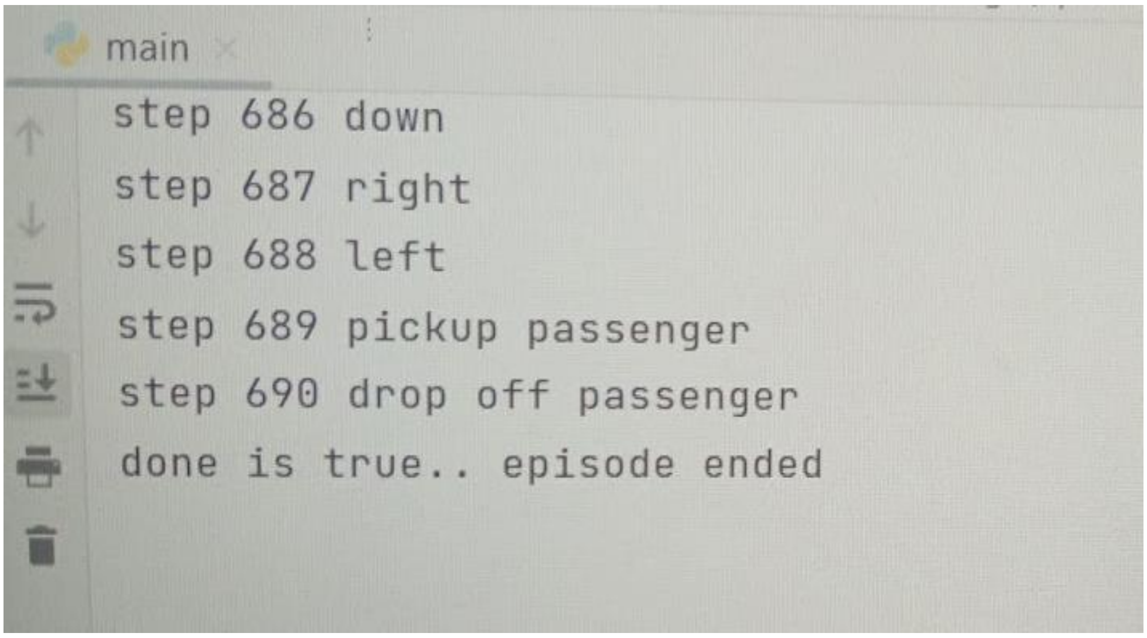
Observations:

When the learning is done for lesser number of steps, the taxi requires more steps for getting to the right spot.

When more learning is done for 1000s of epochs , then we get better results and less steps.

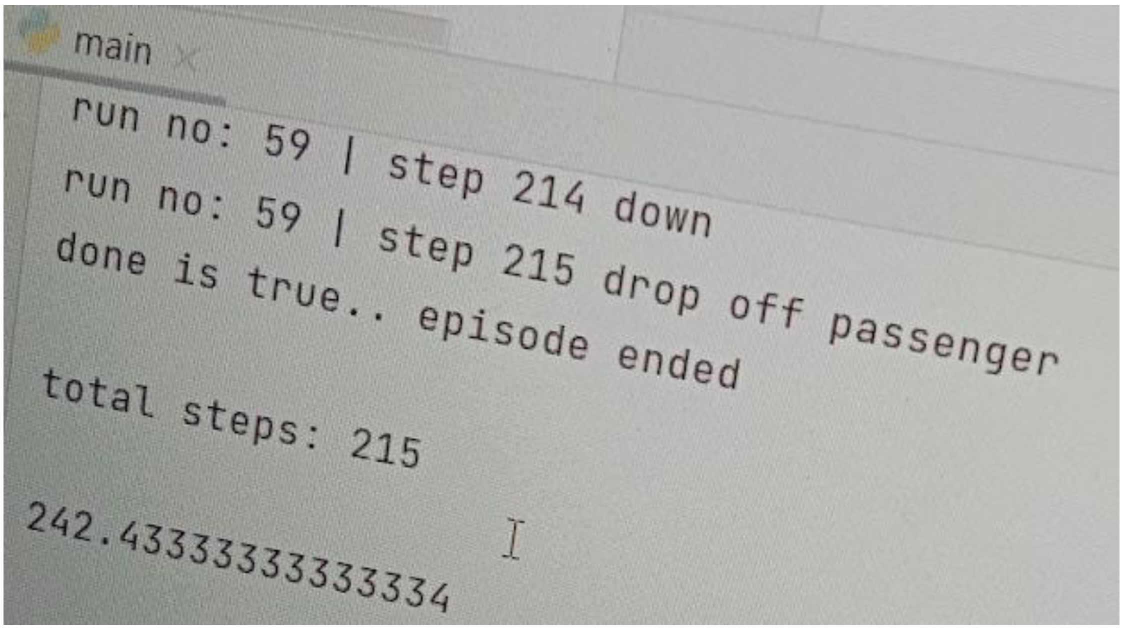
We can also tweak discount rate for getting better results.

1st epoch:



```
step 686 down
step 687 right
step 688 left
step 689 pickup passenger
step 690 drop off passenger
done is true.. episode ended
```

60th epoch: average steps length = 242.43



```
run no: 59 | step 214 down
run no: 59 | step 215 drop off passenger
done is true.. episode ended
total steps: 215
242.43333333333334
```