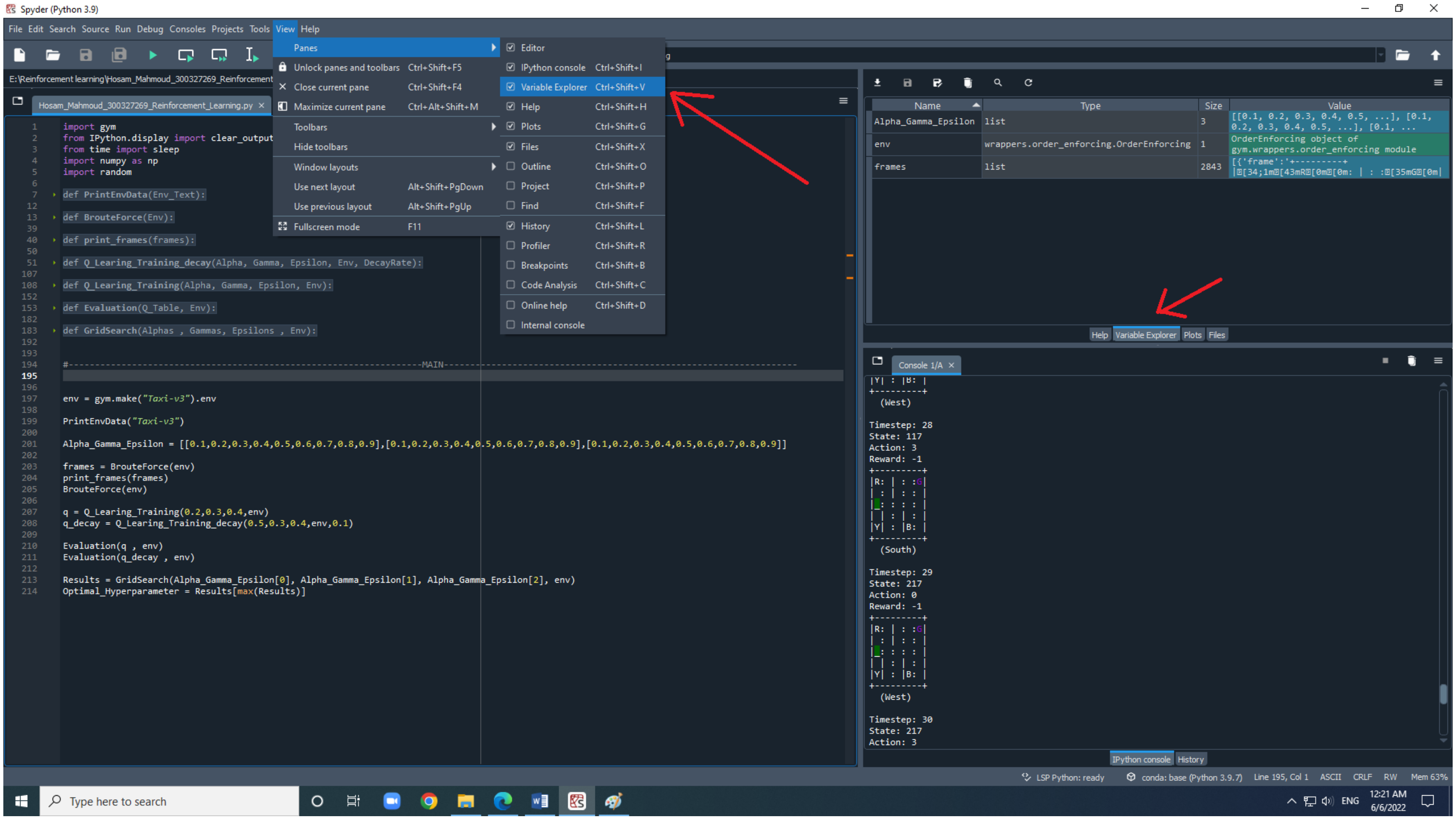
For best experience please consider to run **Hosam_Mahmoud_300327269_Reinforcement_Learning.py** on Spyder to see all created variables on variables explorer tab.


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Reinforcement learning\Hosam_Mahmoud_300327269_Reinforcement
Hosam_Mahmoud_300327269_Reinforcement_Learning.py ×
Panes
Unlock panes and toolbars Ctrl+Shift+F5
Close current pane Ctrl+Shift+F4
Maximize current pane Ctrl+Alt+Shift+M
Toolbars
Hide toolbars
Window layouts
Use next layout Alt+Shift+PgDown
Use previous layout Alt+Shift+PgUp
Fullscreen mode F11
☑ Editor
☑ IPython console Ctrl+Shift+I
☑ Variable Explorer Ctrl+Shift+V
☑ Help Ctrl+Shift+H
☑ Plots Ctrl+Shift+G
☑ Files Ctrl+Shift+X
□ Outline Ctrl+Shift+O
□ Project Ctrl+Shift+P
□ Find Ctrl+Shift+F
☑ History Ctrl+Shift+L
□ Profiler Ctrl+Shift+R
□ Breakpoints Ctrl+Shift+B
□ Code Analysis Ctrl+Shift+C
□ Online help Ctrl+Shift+D
□ Internal console
Help
Variable Explorer
Plots
Files
Console 1/A ×
IPython console
History
LSP Python: ready
conda: base (Python 3.9.7)
Line 195, Col 1
ASCII
CRLF
RW
Mem 63%
Type here to search
ENG
12:21 AM
6/6/2022

```python
import gym
from IPython.display import clear_output
from time import sleep
import numpy as np
import random

def PrintEnvData(Env_Text):

def BrouteForce(Env):

def print_frames(frames):

def Q_Learing_Training_decay(Alpha, Gamma, Epsilon, Env, DecayRate):

def Q_Learing_Training(Alpha, Gamma, Epsilon, Env):

def Evaluation(Q_Table, Env):

def GridSearch(Alphas , Gammas, Epsilons , Env):


#-----------------------------------------------------------------MAIN-----------------------------------------------------------------


env = gym.make("Taxi-v3").env

PrintEnvData("Taxi-v3")

Alpha_Gamma_Epsilon = [[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]]

frames = BrouteForce(env)
print_frames(frames)
BrouteForce(env)

q = Q_Learing_Training(0.2,0.3,0.4,env)
q_decay = Q_Learing_Training_decay(0.5,0.3,0.4,env,0.1)

Evaluation(q , env)
Evaluation(q_decay , env)

Results = GridSearch(Alpha_Gamma_Epsilon[0], Alpha_Gamma_Epsilon[1], Alpha_Gamma_Epsilon[2], env)
Optimal_Hyperparameter = Results[max(Results)]
```

Name	Type	Size	Value
Alpha_Gamma_Epsilon	list	3	[[0.1, 0.2, 0.3, 0.4, 0.5, ...], [0.1, 0.2, 0.3, 0.4, 0.5, ...], [0.1, ...
env	wrappers.order_enforcing.OrderEnforcing	1	OrderEnforcing object of gym.wrappers.order_enforcing module
frames	list	2843	[{'frame':'+---------+ |?[34;1m?[43mR?[0m?[0m: | : :?[35mG?[0m|

```
|Y| : |B: |
+---------+
  (West)

Timestep: 28
State: 117
Action: 3
Reward: -1
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)

Timestep: 29
State: 217
Action: 0
Reward: -1
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)

Timestep: 30
State: 217
Action: 3
```

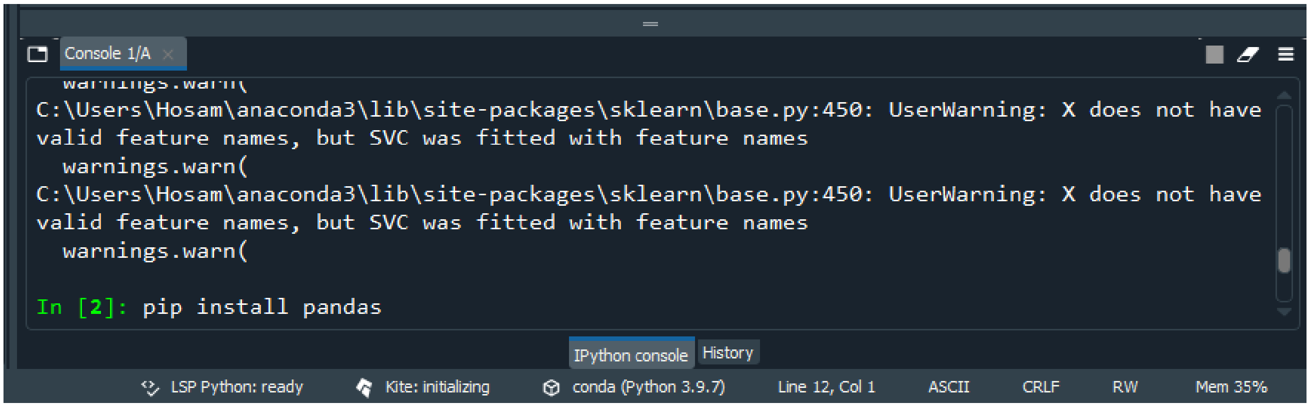
Please make sure that you have installed these libraries before importing it ->

import gym
from IPython.display import clear_output
from time import sleep
import numpy as np
import random

To install the missing libraries, you have to write something like this ->

>>> pip install *numpy*

In Spyder console.

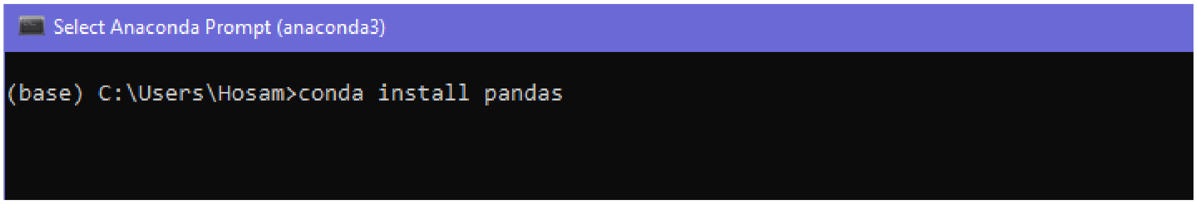


```
  warnings.warn(
C:\Users\Hosam\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have
valid feature names, but SVC was fitted with feature names
  warnings.warn(
C:\Users\Hosam\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have
valid feature names, but SVC was fitted with feature names
  warnings.warn(

In [2]: pip install pandas
```

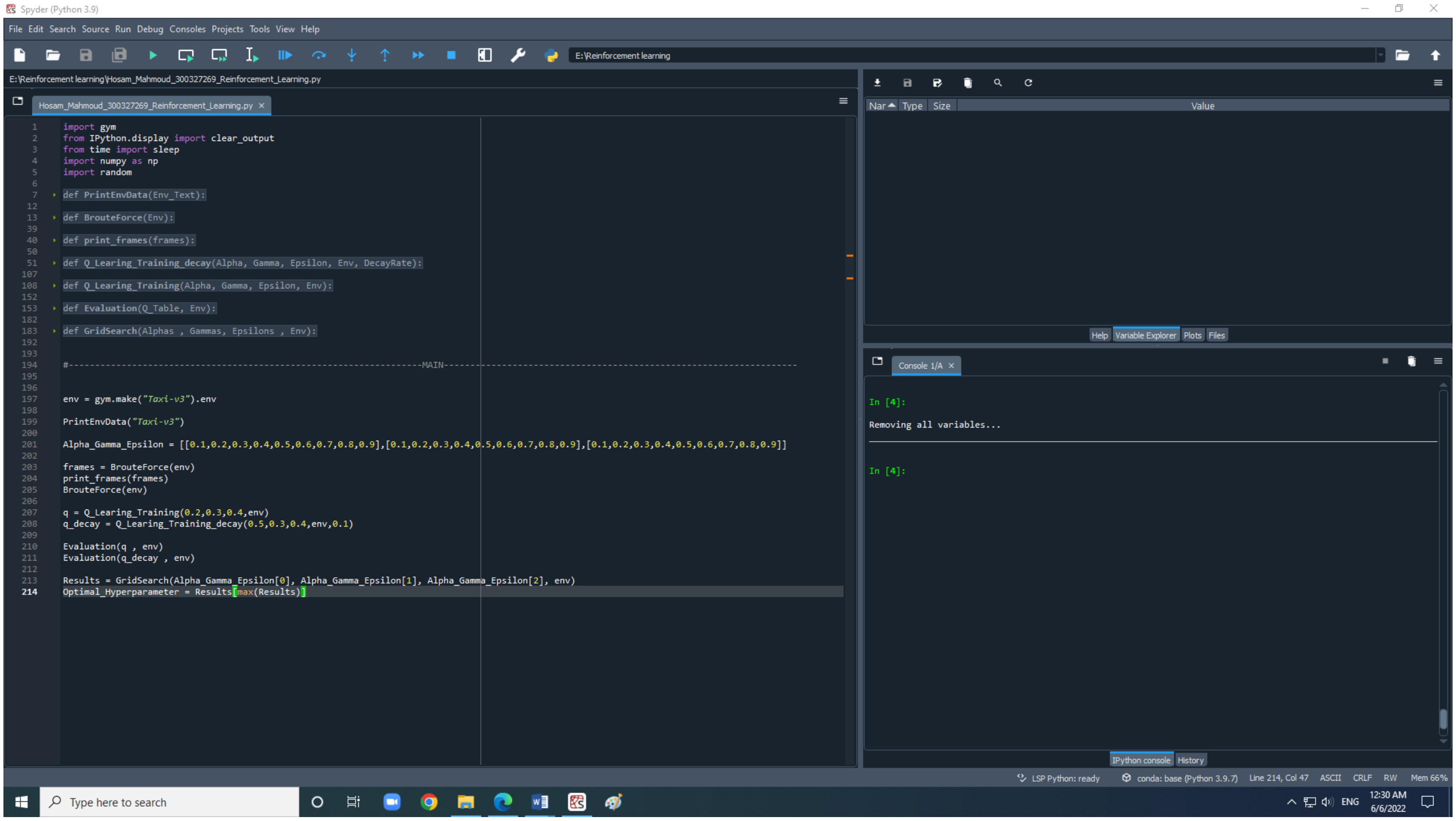
or

>>> conda install pandas

In Anaconda prompt.



```
(base) C:\Users\Hosam>conda install pandas
```



Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Reinforcement learning
E:\Reinforcement learning\Hosam_Mahmoud_300327269_Reinforcement_Learning.py
Hosam_Mahmoud_300327269_Reinforcement_Learning.py
Name Type Size Value
Help Variable Explorer Plots Files
Console 1/A
IPython console History
LSP Python: ready conda: base (Python 3.9.7) Line 214, Col 47 ASCII CRLF RW Mem 66%
Type here to search
ENG 12:30 AM 6/6/2022

```python
import gym
from IPython.display import clear_output
from time import sleep
import numpy as np
import random

def PrintEnvData(Env_Text):

def BrouteForce(Env):

def print_frames(frames):

def Q_Learing_Training_decay(Alpha, Gamma, Epsilon, Env, DecayRate):

def Q_Learing_Training(Alpha, Gamma, Epsilon, Env):

def Evaluation(Q_Table, Env):

def GridSearch(Alphas , Gammas, Epsilons , Env):


#----------------------------------------------------------------------MAIN-------------------------------------------------------------------------


env = gym.make("Taxi-v3").env

PrintEnvData("Taxi-v3")

Alpha_Gamma_Epsilon = [[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]]

frames = BrouteForce(env)
print_frames(frames)
BrouteForce(env)

q = Q_Learing_Training(0.2,0.3,0.4,env)
q_decay = Q_Learing_Training_decay(0.5,0.3,0.4,env,0.1)

Evaluation(q , env)
Evaluation(q_decay , env)

Results = GridSearch(Alpha_Gamma_Epsilon[0], Alpha_Gamma_Epsilon[1], Alpha_Gamma_Epsilon[2], env)
Optimal_Hyperparameter = Results[max(Results)]
```

```
In [4]:

Removing all variables...

In [4]:
```

Here I have 7 functions in my code.

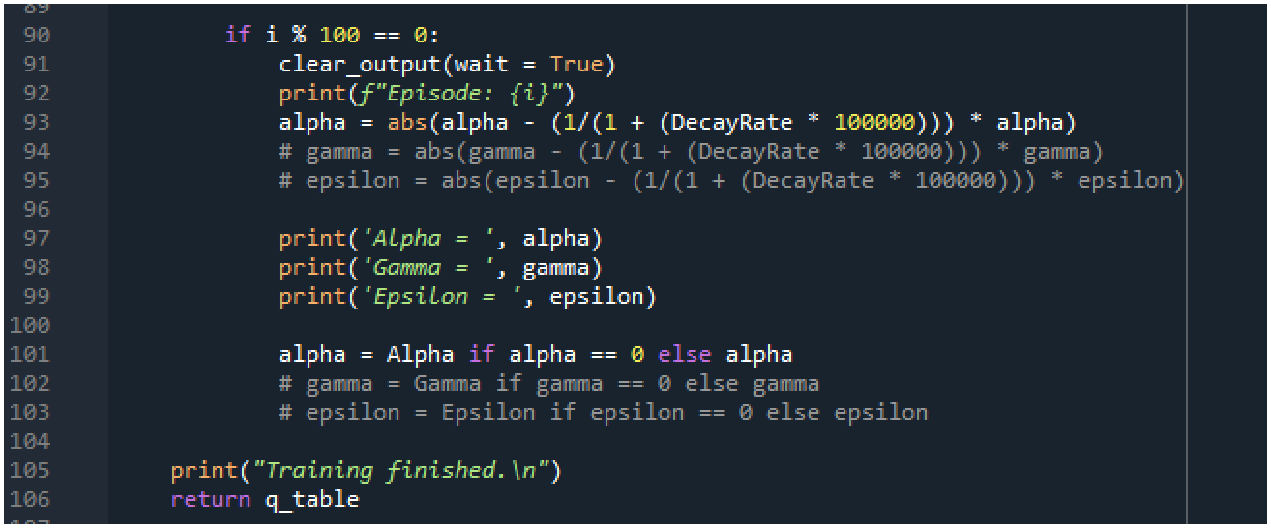
def PrintEnvData(Env_Text): is for printing the number of actions and states of the environments.

def BrouteForce(Env): to try all the possible paths for one state.

def print_frames(frames): to animate the paths.

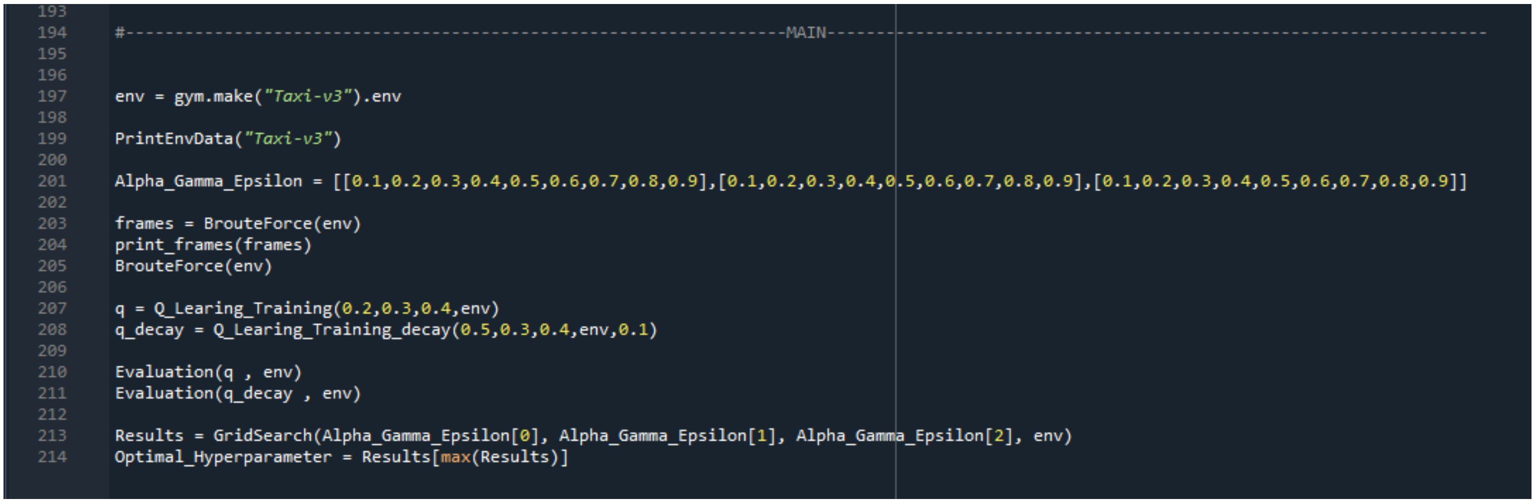
def Q_Learing_Training_decay(Alpha, Gamma, Epsilon, Env, DecayRate): Training function using a decay over episodes.

Here is the parameter tuning part, but in case of **gamma** and **epsilon** I have tried to change its values but I got an infinite loop when applying evaluation.



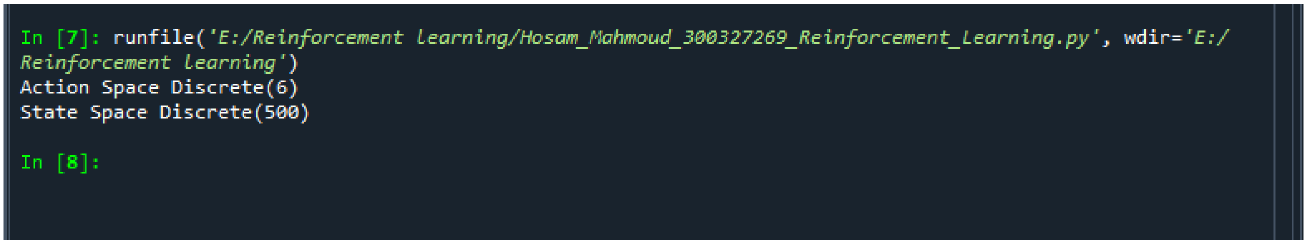
```python
        if i % 100 == 0:
            clear_output(wait = True)
            print(f"Episode: {i}")
            alpha = abs(alpha - (1/(1 + (DecayRate * 100000))) * alpha)
            # gamma = abs(gamma - (1/(1 + (DecayRate * 100000))) * gamma)
            # epsilon = abs(epsilon - (1/(1 + (DecayRate * 100000))) * epsilon)

            print('Alpha = ', alpha)
            print('Gamma = ', gamma)
            print('Epsilon = ', epsilon)

            alpha = Alpha if alpha == 0 else alpha
            # gamma = Gamma if gamma == 0 else gamma
            # epsilon = Epsilon if epsilon == 0 else epsilon

    print("Training finished.\n")
    return q_table
```

def Q_Learing_Training(Alpha, Gamma, Epsilon, Env): Training function without parameter tuning.
def Evaluation(Q_Table, Env): to return the performance indicator of each reinforcement model.
def GridSearch(Alphas , Gammas, Epsilons , Env): to apply GridSearch and optain the optimal hyperparameters.



```

#-------------------------------------------------------------------MAIN--------------------------------------------------------------


env = gym.make("Taxi-v3").env

PrintEnvData("Taxi-v3")

Alpha_Gamma_Epsilon = [[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]]

frames = BrouteForce(env)
print_frames(frames)
BrouteForce(env)

q = Q_Learing_Training(0.2,0.3,0.4,env)
q_decay = Q_Learing_Training_decay(0.5,0.3,0.4,env,0.1)

Evaluation(q , env)
Evaluation(q_decay , env)

Results = GridSearch(Alpha_Gamma_Epsilon[0], Alpha_Gamma_Epsilon[1], Alpha_Gamma_Epsilon[2], env)
Optimal_Hyperparameter = Results[max(Results)]
```

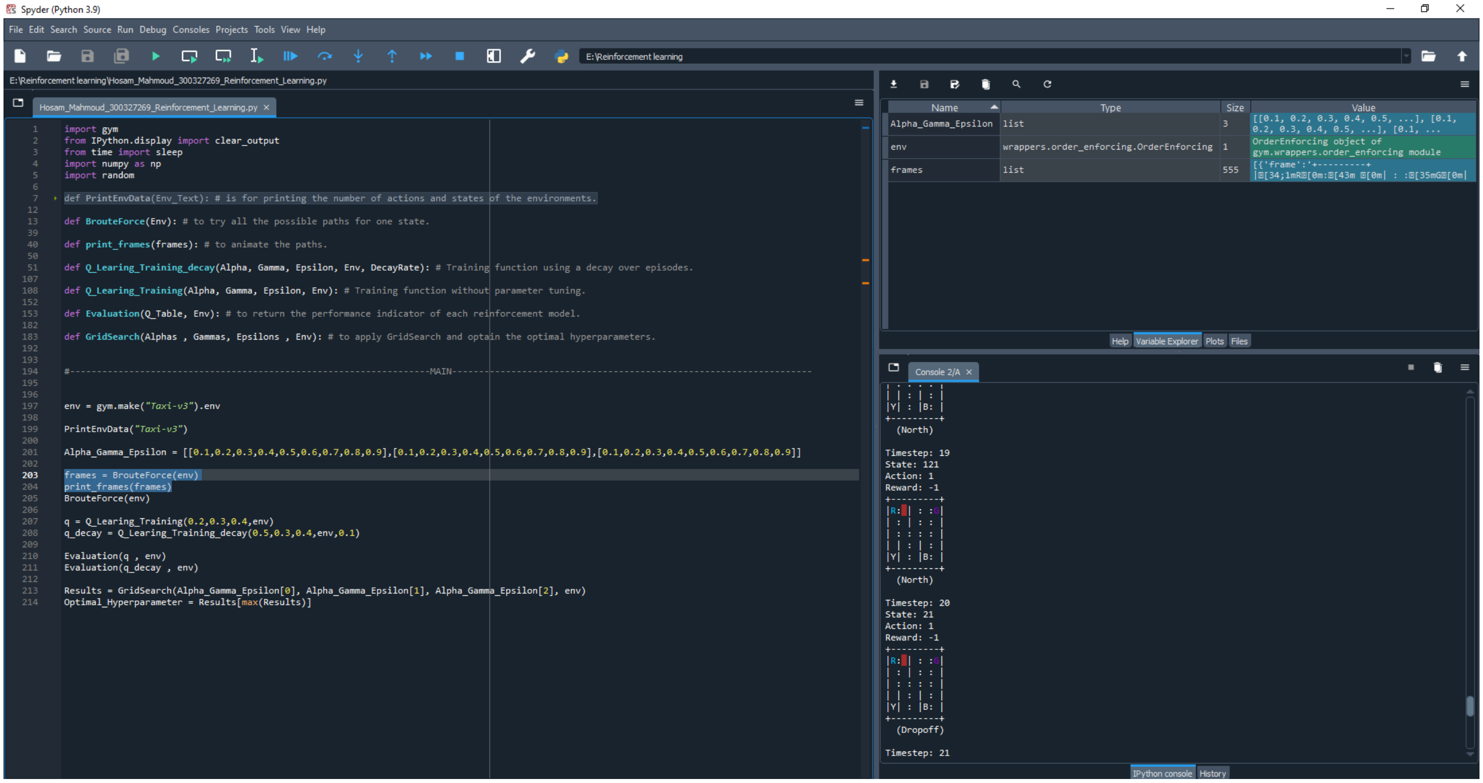
For the MAIN part I have created environment and print its number of states and actions.



```
In [7]: runfile('E:/Reinforcement learning/Hosam_Mahmoud_300327269_Reinforcement_Learning.py', wdir='E:/
Reinforcement learning')
Action Space Discrete(6)
State Space Discrete(500)

In [8]:
```

After that I have created list of each parameter (alpha, gamma and epsilon), and I'll use it for grid search **->** 9*9*9 but this toke too long time, so I tried with another lists **->** [0.3,0.6,0.9] which will give us a total number of combinations = 3*3*3

And after that I animated the paths of Bruteforce function.

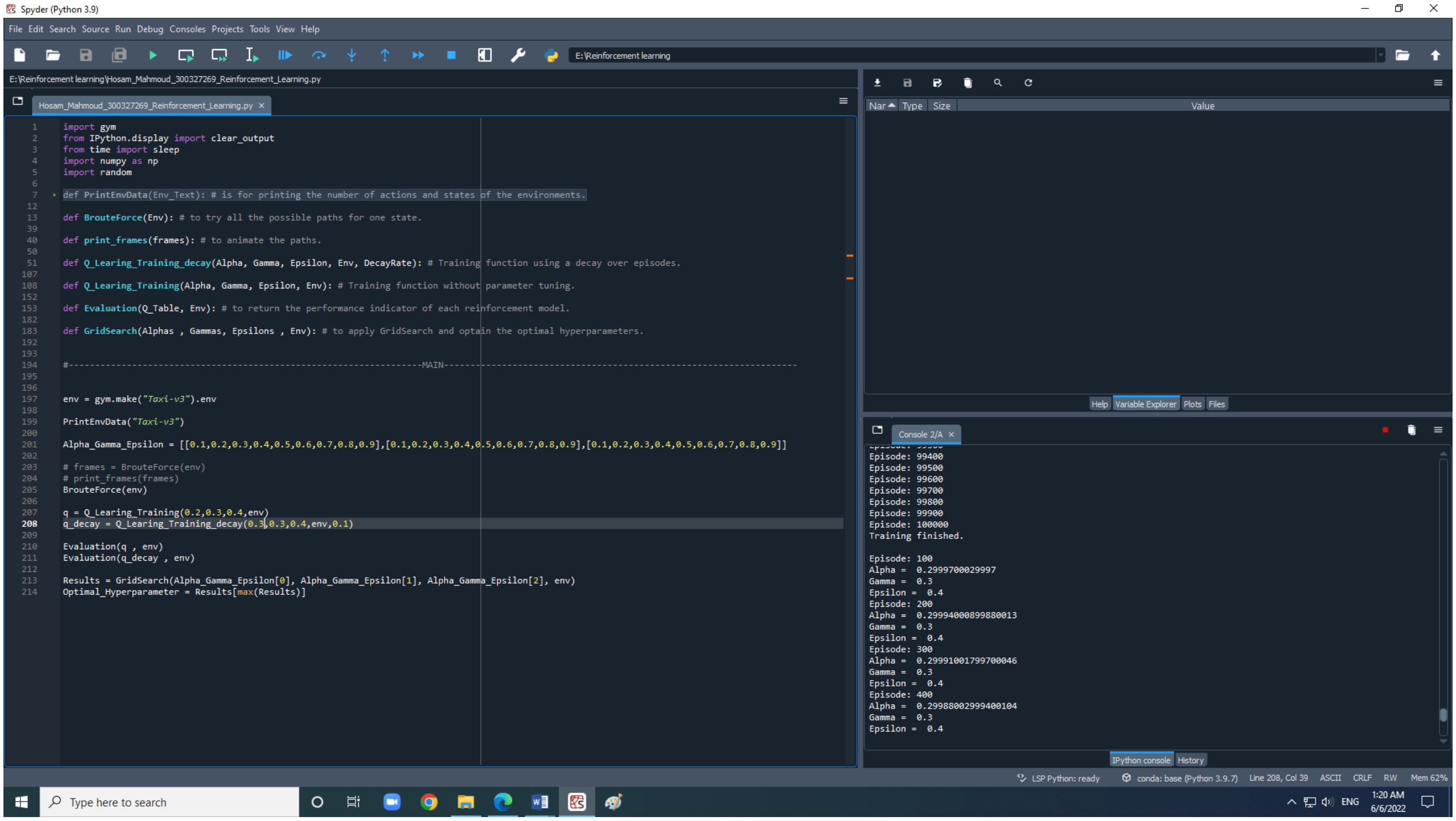

Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Reinforcement learning
E:\Reinforcement learning\Hosam_Mahmoud_300327269_Reinforcement_Learning.py
Hosam_Mahmoud_300327269_Reinforcement_Learning.py
Help Variable Explorer Plots Files
Console 2/A
IPython console History

```python
import gym
from IPython.display import clear_output
from time import sleep
import numpy as np
import random

def PrintEnvData(Env_Text): # is for printing the number of actions and states of the environments.

def BrouteForce(Env): # to try all the possible paths for one state.

def print_frames(frames): # to animate the paths.

def Q_Learing_Training_decay(Alpha, Gamma, Epsilon, Env, DecayRate): # Training function using a decay over episodes.

def Q_Learing_Training(Alpha, Gamma, Epsilon, Env): # Training function without parameter tuning.

def Evaluation(Q_Table, Env): # to return the performance indicator of each reinforcement model.

def GridSearch(Alphas , Gammas, Epsilons , Env): # to apply GridSearch and optain the optimal hyperparameters.


#-----------------------------------------------------------------MAIN------------------------------------------------------------------


env = gym.make("Taxi-v3").env

PrintEnvData("Taxi-v3")

Alpha_Gamma_Epsilon = [[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]]

frames = BrouteForce(env)
print_frames(frames)
BrouteForce(env)

q = Q_Learing_Training(0.2,0.3,0.4,env)
q_decay = Q_Learing_Training_decay(0.5,0.3,0.4,env,0.1)

Evaluation(q , env)
Evaluation(q_decay , env)

Results = GridSearch(Alpha_Gamma_Epsilon[0], Alpha_Gamma_Epsilon[1], Alpha_Gamma_Epsilon[2], env)
Optimal_Hyperparameter = Results[max(Results)]
```

Name	Type	Size	Value
Alpha_Gamma_Epsilon	list	3	[[0.1, 0.2, 0.3, 0.4, 0.5, ...], [0.1, 0.2, 0.3, 0.4, 0.5, ...], [0.1, ...
env	wrappers.order_enforcing.OrderEnforcing	1	OrderEnforcing object of gym.wrappers.order_enforcing module
frames	list	555	[{'frame':'+---------+ |⊠[34;1mR⊠[0m:⊠[43m ⊠[0m| : :⊠[35mG⊠[0m|

```
| | : | : |
|Y| : |B: |
+---------+
  (North)

Timestep: 19
State: 121
Action: 1
Reward: -1
+---------+
|R:█| : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)

Timestep: 20
State: 21
Action: 1
Reward: -1
+---------+
|R:█| : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Timestep: 21
```

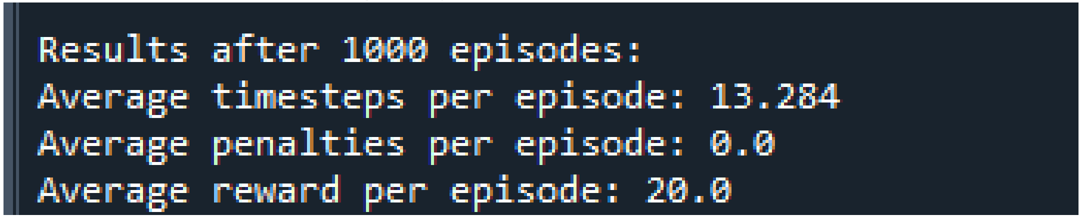
After that I trained the two models (using decay and without using decay). And the values of alpha are changed every 100 episodes.


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Reinforcement learning
E:\Reinforcement learning\Hosam_Mahmoud_300327269_Reinforcement_Learning.py
Hosam_Mahmoud_300327269_Reinforcement_Learning.py
Nar Type Size Value
Help Variable Explorer Plots Files
Console 2/A
IPython console History
LSP Python: ready
conda: base (Python 3.9.7)
Line 208, Col 39
ASCII
CRLF
RW
Mem 62%
Type here to search
ENG
1:20 AM
6/6/2022

```python
import gym
from IPython.display import clear_output
from time import sleep
import numpy as np
import random

def PrintEnvData(Env_Text): # is for printing the number of actions and states of the environments.

def BrouteForce(Env): # to try all the possible paths for one state.

def print_frames(frames): # to animate the paths.

def Q_Learing_Training_decay(Alpha, Gamma, Epsilon, Env, DecayRate): # Training function using a decay over episodes.

def Q_Learing_Training(Alpha, Gamma, Epsilon, Env): # Training function without parameter tuning.

def Evaluation(Q_Table, Env): # to return the performance indicator of each reinforcement model.

def GridSearch(Alphas , Gammas, Epsilons , Env): # to apply GridSearch and optain the optimal hyperparameters.


#-----------------------------------------------------------------MAIN------------------------------------------------------------------


env = gym.make("Taxi-v3").env

PrintEnvData("Taxi-v3")

Alpha_Gamma_Epsilon = [[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]]

# frames = BrouteForce(env)
# print_frames(frames)
BrouteForce(env)

q = Q_Learing_Training(0.2,0.3,0.4,env)
q_decay = Q_Learing_Training_decay(0.3,0.3,0.4,env,0.1)

Evaluation(q , env)
Evaluation(q_decay , env)

Results = GridSearch(Alpha_Gamma_Epsilon[0], Alpha_Gamma_Epsilon[1], Alpha_Gamma_Epsilon[2], env)
Optimal_Hyperparameter = Results[max(Results)]
```

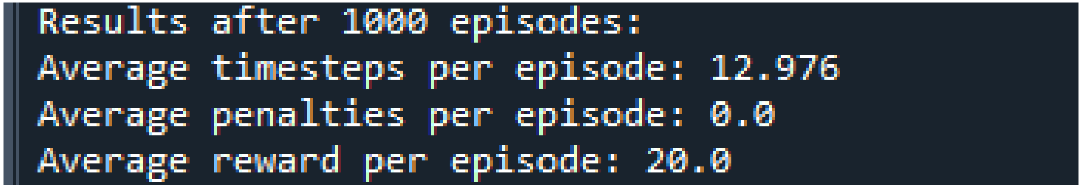
```
Episode: 99400
Episode: 99500
Episode: 99600
Episode: 99700
Episode: 99800
Episode: 99900
Episode: 100000
Training finished.

Episode: 100
Alpha =  0.2999700029997
Gamma =  0.3
Epsilon =  0.4
Episode: 200
Alpha =  0.29994000899880013
Gamma =  0.3
Epsilon =  0.4
Episode: 300
Alpha =  0.29991001799700046
Gamma =  0.3
Epsilon =  0.4
Episode: 400
Alpha =  0.29988002999400104
Gamma =  0.3
Epsilon =  0.4
```

After Evaluation (without using decay)



```
Results after 1000 episodes:
Average timesteps per episode: 13.284
Average penalties per episode: 0.0
Average reward per episode: 20.0
```

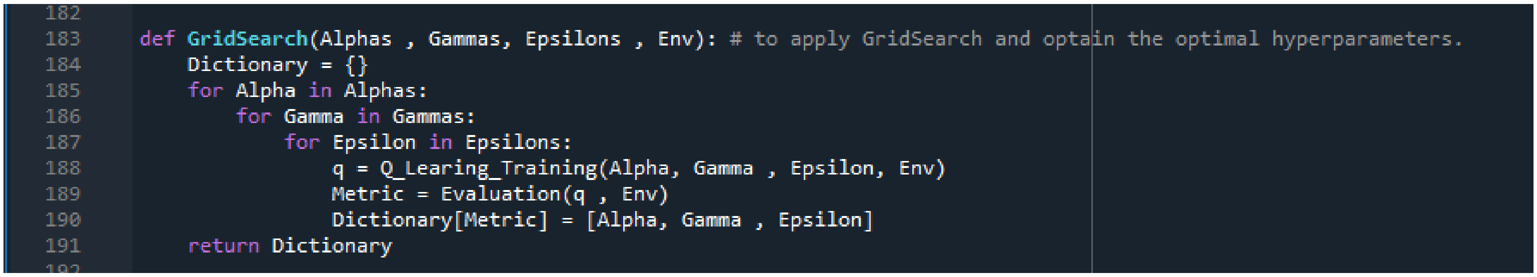
After Evaluation (using decay)



```
Results after 1000 episodes:
Average timesteps per episode: 12.976
Average penalties per episode: 0.0
Average reward per episode: 20.0
```

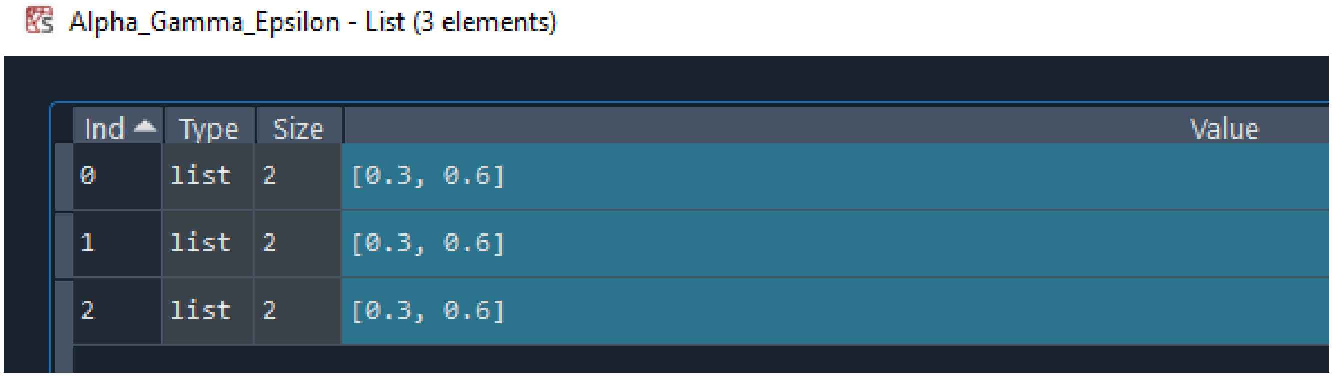
And we found that (with using decay) is faster than the normal model.

After that I applied GridSearch on function (without using Decay)



```python
def GridSearch(Alphas , Gammas, Epsilons , Env): # to apply GridSearch and optain the optimal hyperparameters.
    Dictionary = {}
    for Alpha in Alphas:
        for Gamma in Gammas:
            for Epsilon in Epsilons:
                q = Q_Learing_Training(Alpha, Gamma , Epsilon, Env)
                Metric = Evaluation(q , Env)
                Dictionary[Metric] = [Alpha, Gamma , Epsilon]
    return Dictionary
```

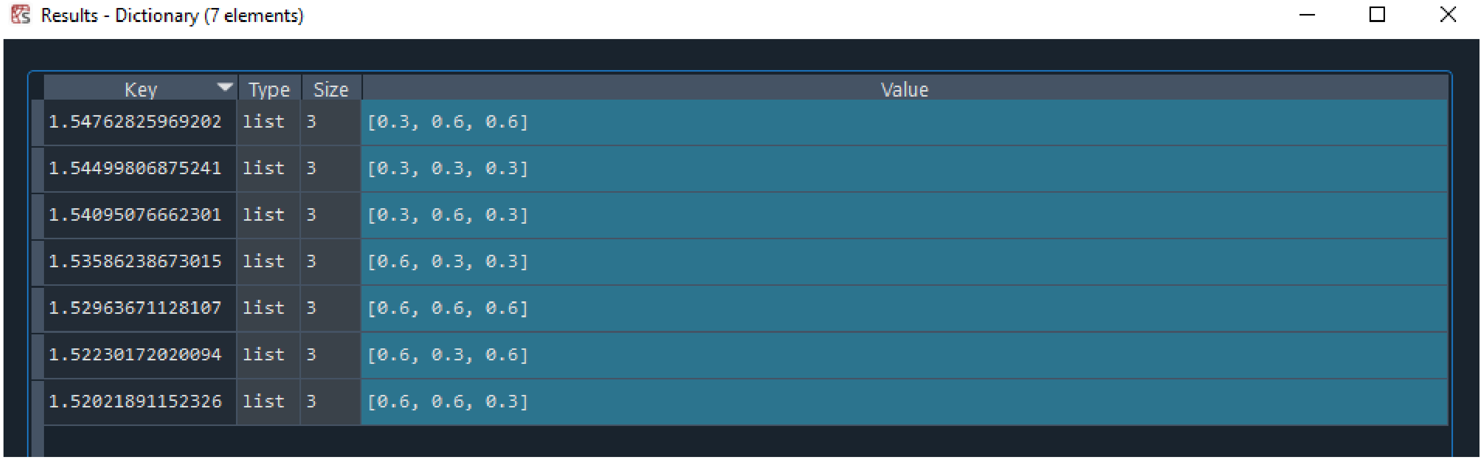
For experiments purposes I have choose small values for parameters so the program run faster -> 2*2*2



Alpha_Gamma_Epsilon - List (3 elements)

Ind	Type	Size	Value
0	list	2	[0.3, 0.6]
1	list	2	[0.3, 0.6]
2	list	2	[0.3, 0.6]

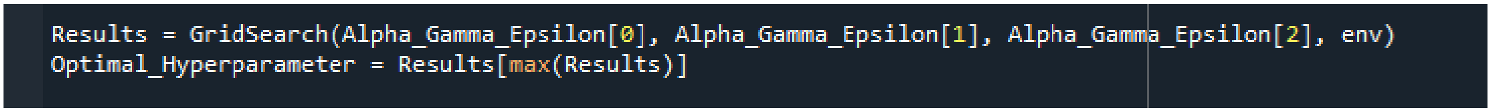
After applying GridSearch we will choose the maximum key and return its parameter And the key is equal to that equation -> **Metric = (total_reward/(total_penalties + total_epochs)).**



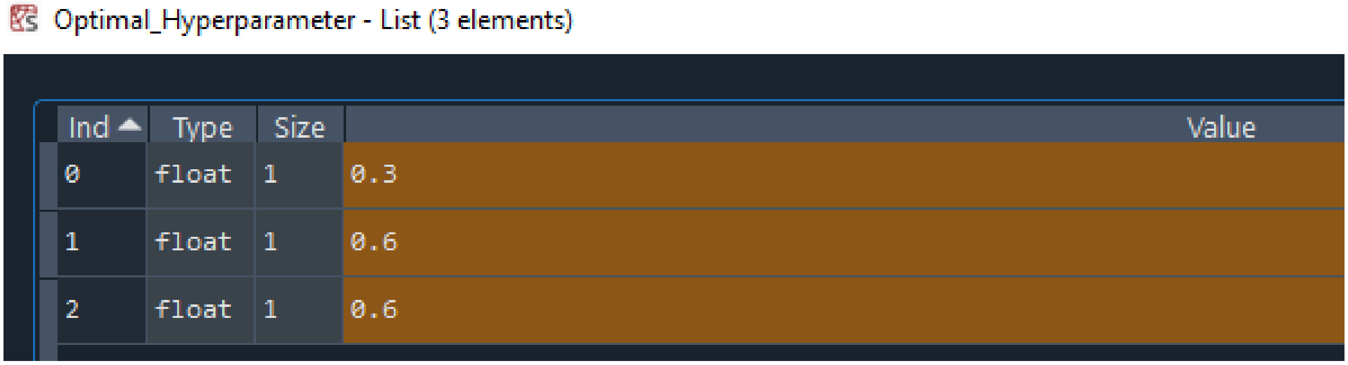
Results - Dictionary (7 elements)

Key	Type	Size	Value
1.54762825969202	list	3	[0.3, 0.6, 0.6]
1.54499806875241	list	3	[0.3, 0.3, 0.3]
1.54095076662301	list	3	[0.3, 0.6, 0.3]
1.53586238673015	list	3	[0.6, 0.3, 0.3]
1.52963671128107	list	3	[0.6, 0.6, 0.6]
1.52230172020094	list	3	[0.6, 0.3, 0.6]
1.52021891152326	list	3	[0.6, 0.6, 0.3]

And Finally, we obtained the optimal hyperparameters.



```
Results = GridSearch(Alpha_Gamma_Epsilon[0], Alpha_Gamma_Epsilon[1], Alpha_Gamma_Epsilon[2], env)
Optimal_Hyperparameter = Results[max(Results)]
```



Optimal_Hyperparameter - List (3 elements)

Ind	Type	Size	Value
0	float	1	0.3
1	float	1	0.6
2	float	1	0.6