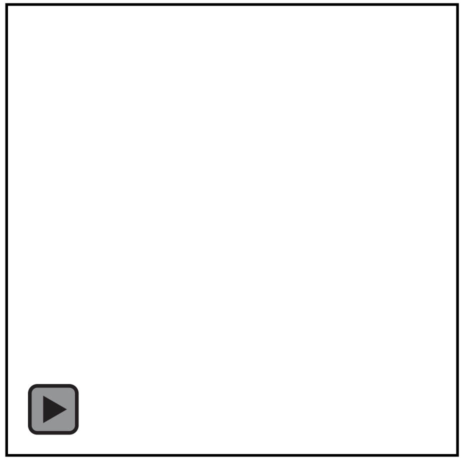
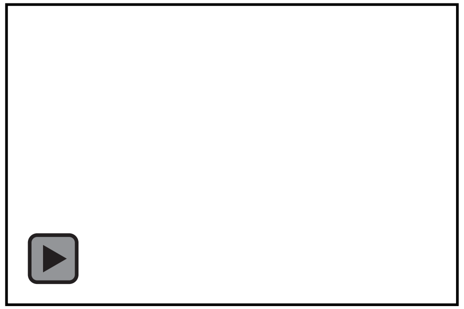
Behavior Cloning

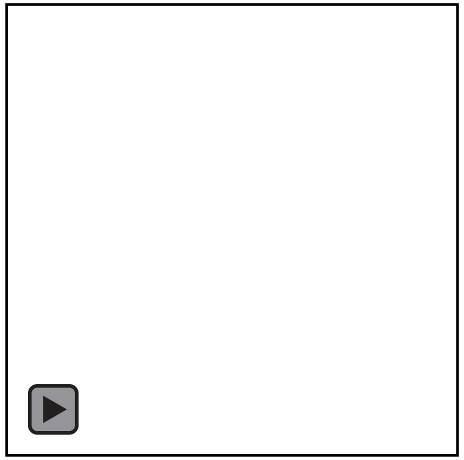
Kyungjae Lee, Yunho Choi, Timothy Ha
RLLAB

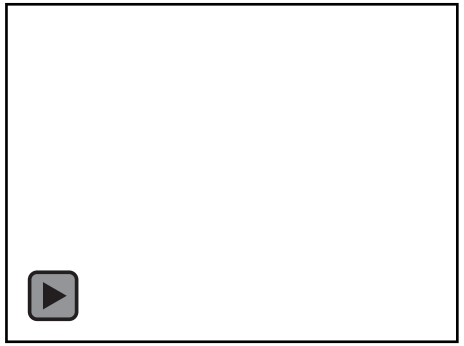
[Exercise 1] Gym Environment

- Behavior Cloning Problem
  - Given control data $\{s, a\}$
  - Find controller from data (Supervised learning!)
  - Today, we will use a Gaussian process regression (GPR)
- Gym Environment









[Exercise 1] Gym Environment

- Install gym Environment
  - Pip install gym

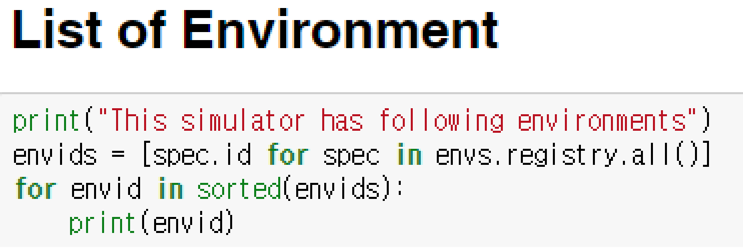
- Install JSAnimation
  - git clone https://github.com/jakevdp/JSAnimation
  - cd JSAnimation
  - python setup.py install

[Exercise 1] Gym Environment

- Gym Environment
  - Import gym

```
import gym
```

  - Gym has many simulators



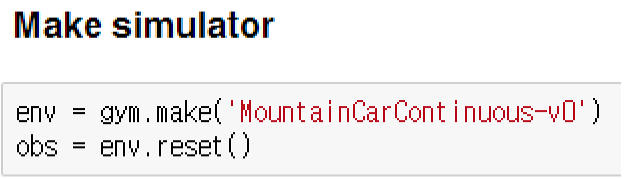
List of Environment

```
print("This simulator has following environments")
envids = [spec.id for spec in envs.registry.all()]
for envid in sorted(envids):
    print(envid)
```

  - We will use MountainCar and Penulum

[Exercise 1] Gym Environment

- Mountain Car Environment

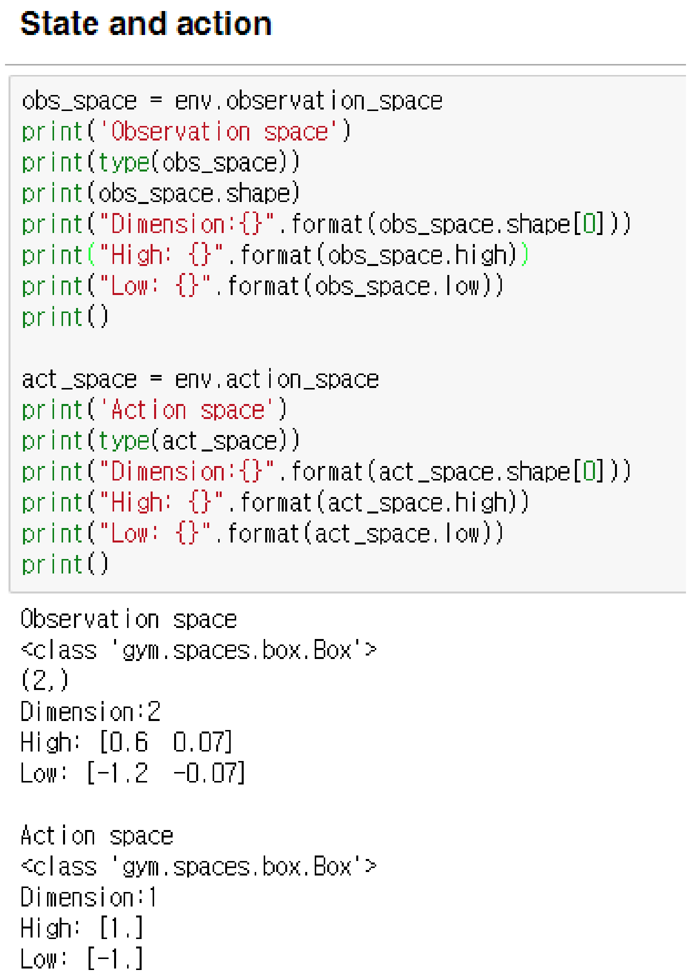


Make simulator

```
env = gym.make('MountainCarContinuous-v0')
obs = env.reset()
```

Continuous state
3 dimension

Continuous action
1 dimension



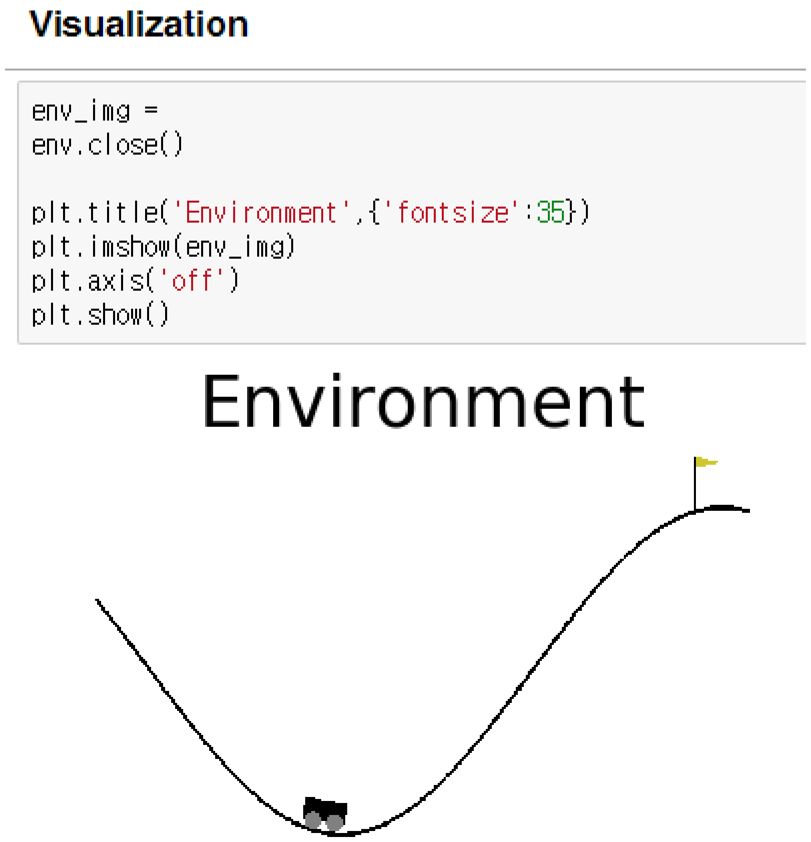
State and action

```
obs_space = env.observation_space
print('Observation space')
print(type(obs_space))
print(obs_space.shape)
print("Dimension:{}".format(obs_space.shape[0]))
print("High: {}".format(obs_space.high))
print("Low: {}".format(obs_space.low))
print()

act_space = env.action_space
print('Action space')
print(type(act_space))
print("Dimension:{}".format(act_space.shape[0]))
print("High: {}".format(act_space.high))
print("Low: {}".format(act_space.low))
print()
```

```
Observation space
<class 'gym.spaces.box.Box'>
(2,)
Dimension:2
High: [0.6  0.07]
Low: [-1.2  -0.07]

Action space
<class 'gym.spaces.box.Box'>
Dimension:1
High: [1.]
Low: [-1.]
```

[Exercise 1] Gym Environment

- Mountain Car Environment
  - Visualize simulation
  - env.render(mode = 'rgb_array')



Visualization

```
env_img =
env.close()

plt.title('Environment',{'fontsize':35})
plt.imshow(env_img)
plt.axis('off')
plt.show()
```

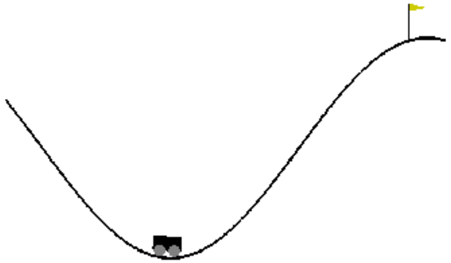
[Exercise 1] Gym Environment

- Mountain Car Environment
  - Control the agent using step() function

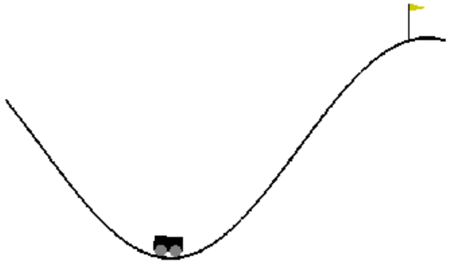
  - action = env.action_space.sample() (Random action sampler)
  - env.step(action)

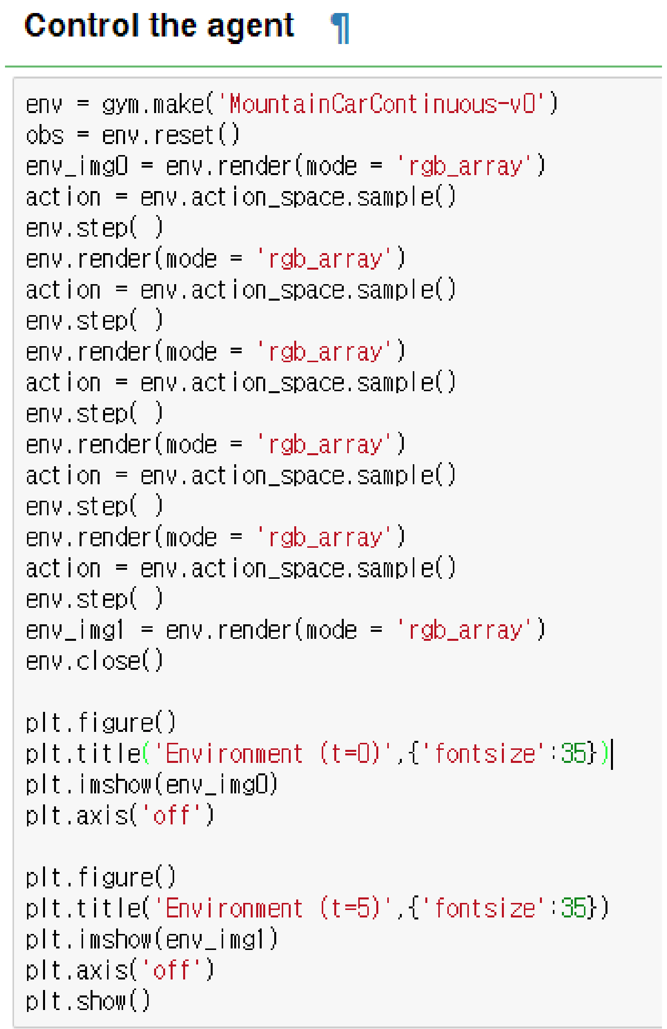
  - When using GPR controller
  - ation = GPR(state)

Environment (t=0)



Environment (t=5)



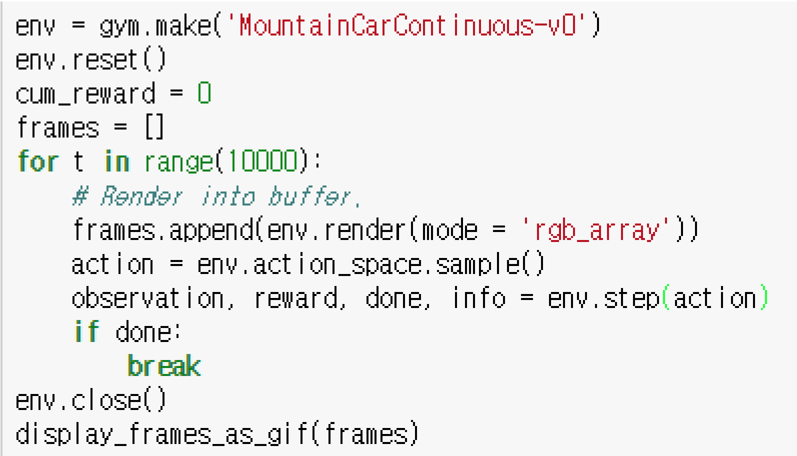


Control the agent ¶

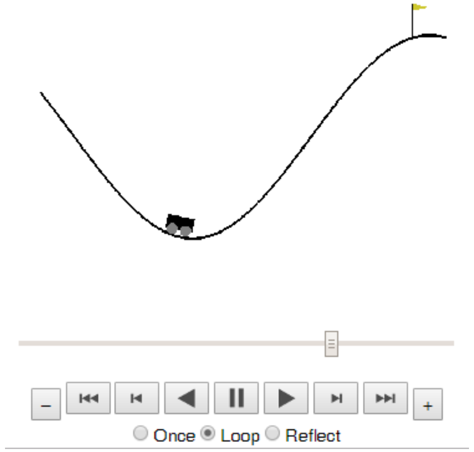
```
env = gym.make('MountainCarContinuous-v0')
obs = env.reset()
env_img0 = env.render(mode = 'rgb_array')
action = env.action_space.sample()
env.step( )
env.render(mode = 'rgb_array')
action = env.action_space.sample()
env.step( )
env.render(mode = 'rgb_array')
action = env.action_space.sample()
env.step( )
env.render(mode = 'rgb_array')
action = env.action_space.sample()
env.step( )
env.render(mode = 'rgb_array')
action = env.action_space.sample()
env.step( )
env_img1 = env.render(mode = 'rgb_array')
env.close()

plt.figure()
plt.title('Environment (t=0)',{'fontsize':35})
plt.imshow(env_img0)
plt.axis('off')

plt.figure()
plt.title('Environment (t=5)',{'fontsize':35})
plt.imshow(env_img1)
plt.axis('off')
plt.show()
```

[Exercise 1] Gym Environment

- Mountain Car Environment
  - Output of step function is as follows
    - Next observation
    - Reward
    - Done
    - Info
  - Play it



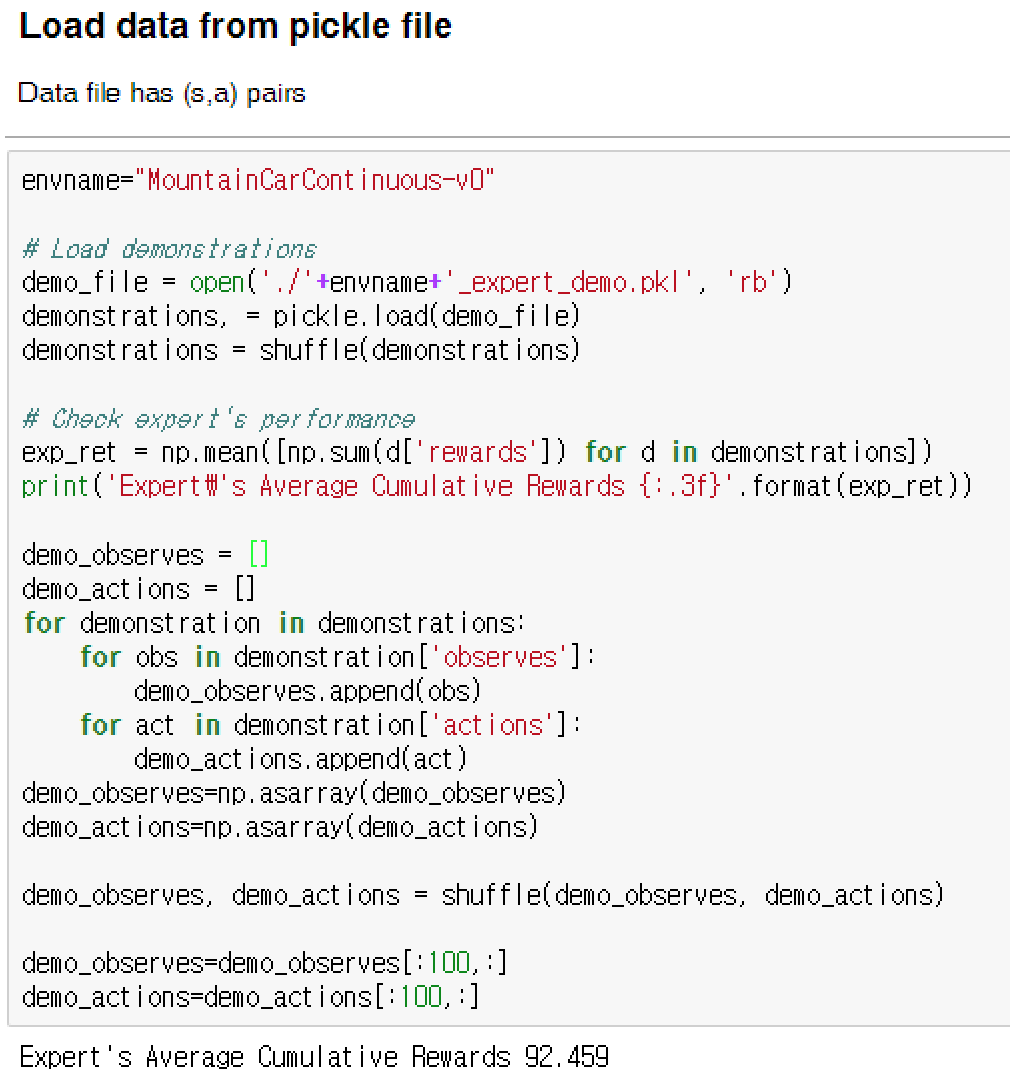
```
env = gym.make('MountainCarContinuous-v0')
env.reset()
cum_reward = 0
frames = []
for t in range(10000):
    # Render into buffer.
    frames.append(env.render(mode = 'rgb_array'))
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
    if done:
        break
env.close()
display_frames_as_gif(frames)
```



[Exercise 2] Behavior Cloning with GPR

- We provide you control data
  - demo_observes : input data
  - 100 by 3 matrix

  - demo_actions : output data
  - 100 by 1 matrix



Load data from pickle file

Data file has (s,a) pairs

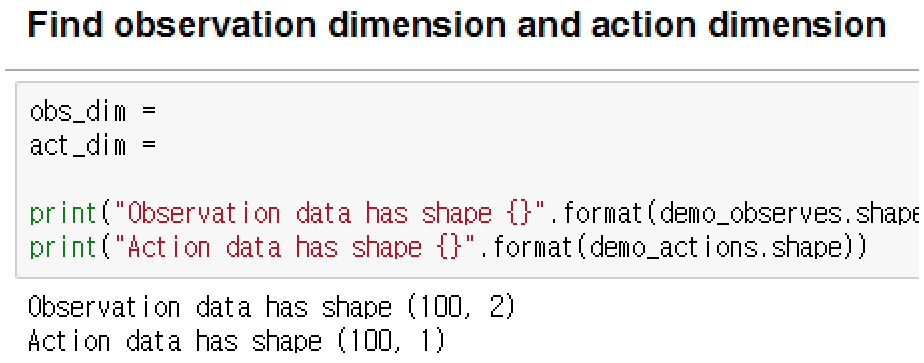
```
envname="MountainCarContinuous-v0"

# Load demonstrations
demo_file = open('./'+envname+'_expert_demo.pkl', 'rb')
demonstrations, = pickle.load(demo_file)
demonstrations = shuffle(demonstrations)

# Check expert's performance
exp_ret = np.mean([np.sum(d['rewards']) for d in demonstrations])
print('Expert₩'s Average Cumulative Rewards {:.3f}'.format(exp_ret))

demo_observes = []
demo_actions = []
for demonstration in demonstrations:
    for obs in demonstration['observes']:
        demo_observes.append(obs)
    for act in demonstration['actions']:
        demo_actions.append(act)
demo_observes=np.asarray(demo_observes)
demo_actions=np.asarray(demo_actions)

demo_observes, demo_actions = shuffle(demo_observes, demo_actions)

demo_observes=demo_observes[:100,:]
demo_actions=demo_actions[:100,:]
```

```
Expert's Average Cumulative Rewards 92.459
```

[Exercise 2] Behavior Cloning with GPR

- Find dimension of state and action
  - Obs_dim = demo_observes.shape[1]
  - Act_dim = demo_actions.shape[1]

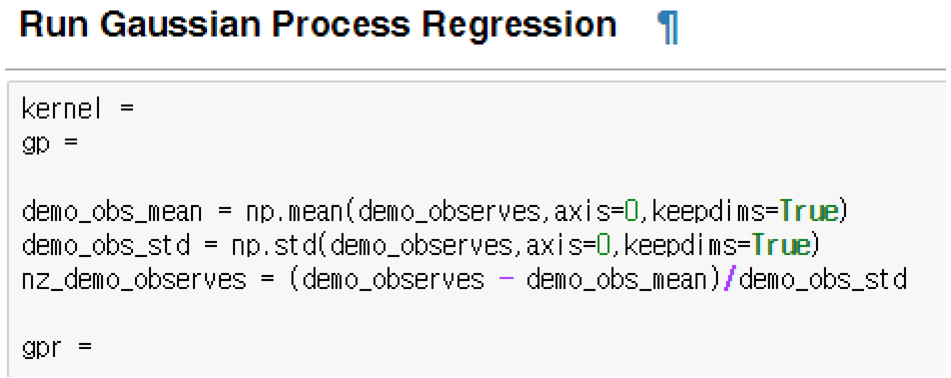
- Define kernel and GP using sklearn
  - kernel = C(1.0, (1e-3, 1e3)) * RBF(1, (1e-2, 1e2))
  - gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
  - gpr = gp.fit(nz_demo_observes, demo_actions)



Find observation dimension and action dimension

```
obs_dim =
act_dim =

print("Observation data has shape {}".format(demo_observes.shape
print("Action data has shape {}".format(demo_actions.shape))
```

```
Observation data has shape (100, 2)
Action data has shape (100, 1)
```

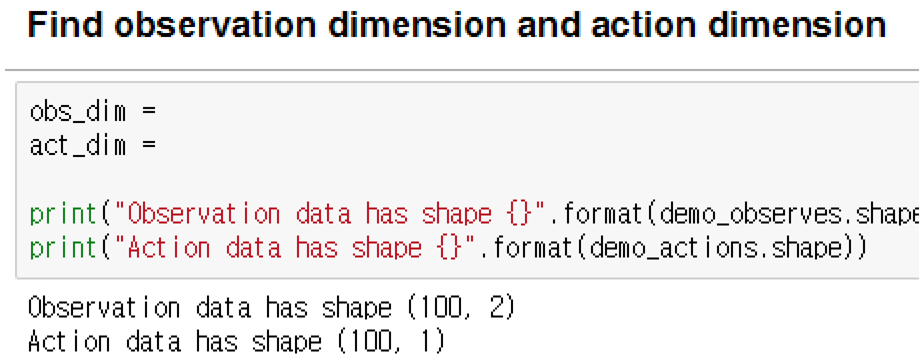


Run Gaussian Process Regression ¶

```
kernel =
gp =

demo_obs_mean = np.mean(demo_observes,axis=0,keepdims=True)
demo_obs_std = np.std(demo_observes,axis=0,keepdims=True)
nz_demo_observes = (demo_observes - demo_obs_mean)/demo_obs_std

gpr =
```

[Exercise 2] Behavior Cloning with GPR

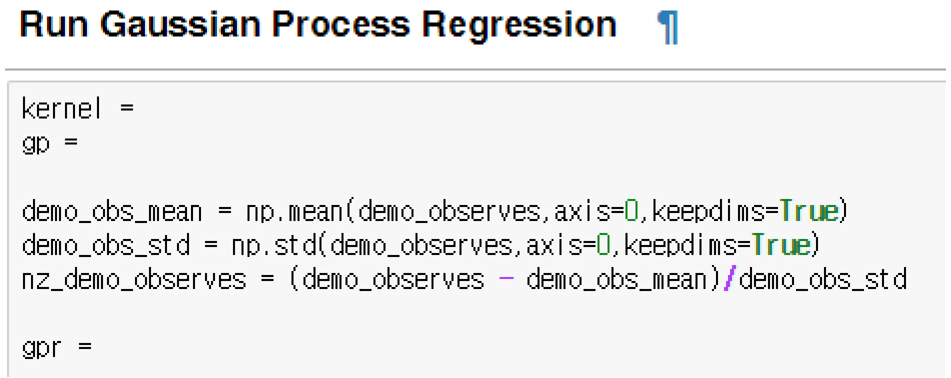
- Find dimension of state and action
  - Obs_dim = demo_observes.shape[1]
  - Act_dim = demo_actions.shape[1]
- Define kernel and GP using sklearn
  - kernel = C(1.0, (1e-3, 1e3)) * RBF(1, (1e-2, 1e2))
  - gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
  - gpr = gp.fit(nz_demo_observes, demo_actions)



Find observation dimension and action dimension

```
obs_dim =
act_dim =

print("Observation data has shape {}".format(demo_observes.shape
print("Action data has shape {}".format(demo_actions.shape))
```

```
Observation data has shape (100, 2)
Action data has shape (100, 1)
```

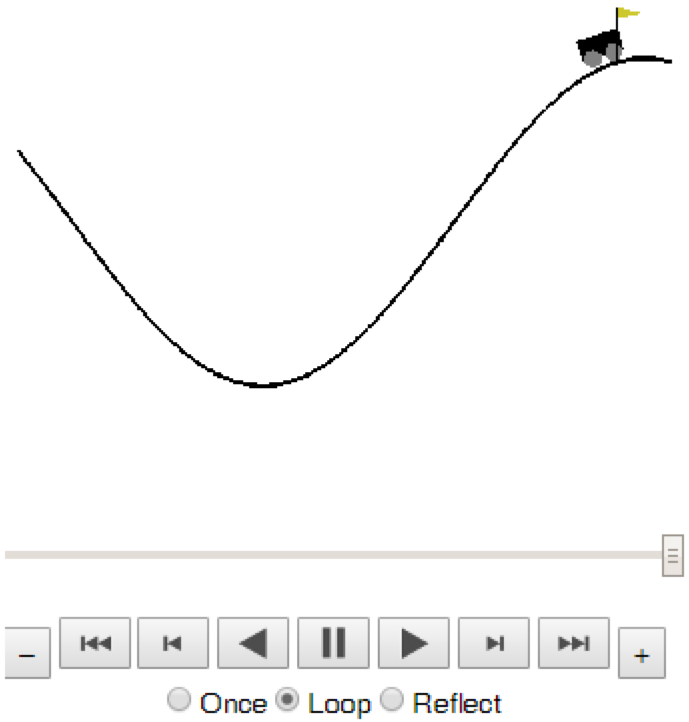


Run Gaussian Process Regression ¶

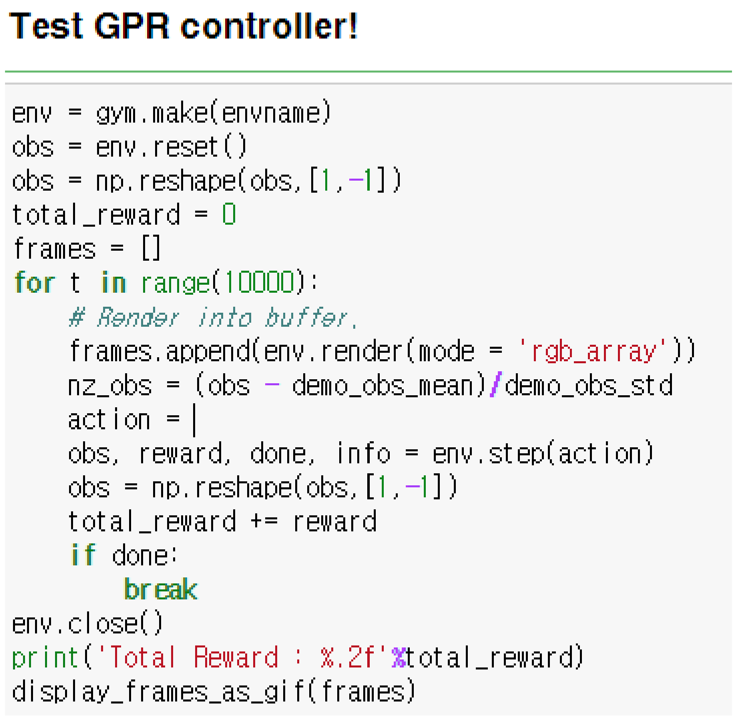
```
kernel =
gp =

demo_obs_mean = np.mean(demo_observes,axis=0,keepdims=True)
demo_obs_std = np.std(demo_observes,axis=0,keepdims=True)
nz_demo_observes = (demo_observes - demo_obs_mean)/demo_obs_std

gpr =
```

[Exercise 2] Behavior Cloning with GPR

- Test GPR controller
  - action = gp.predict(nz_obs)



○ Once ● Loop ○ Reflect

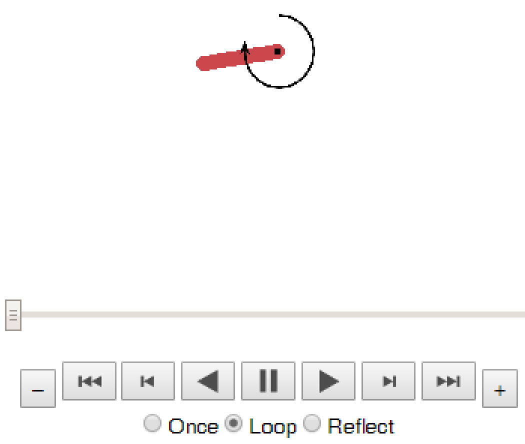


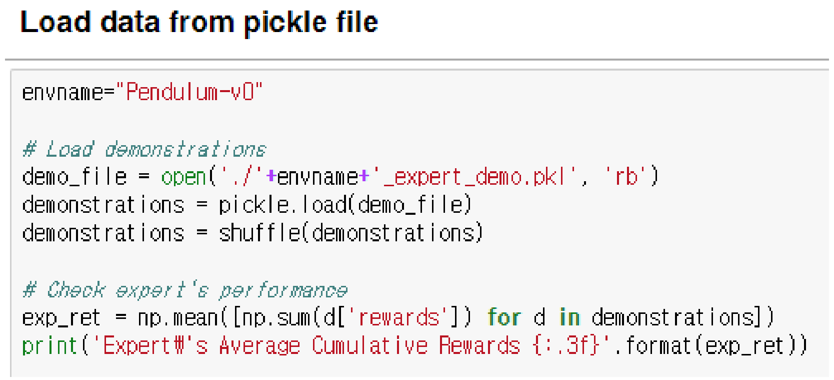
Test GPR controller!

```
env = gym.make(envname)
obs = env.reset()
obs = np.reshape(obs,[1,-1])
total_reward = 0
frames = []
for t in range(10000):
    # Render into buffer.
    frames.append(env.render(mode = 'rgb_array'))
    nz_obs = (obs - demo_obs_mean)/demo_obs_std
    action = 
    obs, reward, done, info = env.step(action)
    obs = np.reshape(obs,[1,-1])
    total_reward += reward
    if done:
        break
env.close()
print('Total Reward : %.2f'%total_reward)
display_frames_as_gif(frames)
```

[Exercise 3] Solve Pendulum-v0

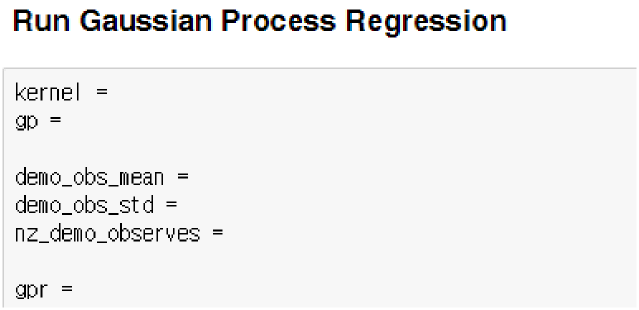
- Load data
- Define GPR using sklearn
- Fit GPR hyper parameters
- Run the trained controller





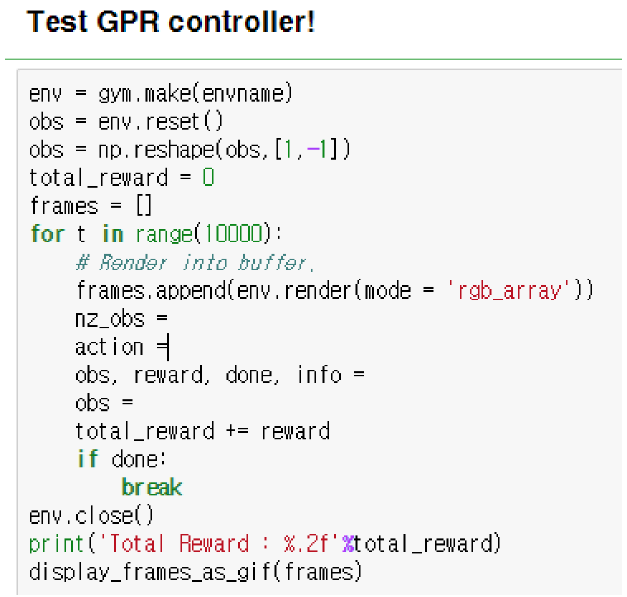
Load data from pickle file

```
envname="Pendulum-v0"

# Load demonstrations
demo_file = open('./'+envname+'_expert_demo.pkl', 'rb')
demonstrations = pickle.load(demo_file)
demonstrations = shuffle(demonstrations)

# Check expert's performance
exp_ret = np.mean([np.sum(d['rewards']) for d in demonstrations])
print('Expert₩'s Average Cumulative Rewards {:.3f}'.format(exp_ret))
```



Run Gaussian Process Regression

```
kernel =
gp =

demo_obs_mean =
demo_obs_std =
nz_demo_observes =

gpr =
```



Test GPR controller!

```
env = gym.make(envname)
obs = env.reset()
obs = np.reshape(obs,[1,-1])
total_reward = 0
frames = []
for t in range(10000):
    # Render into buffer.
    frames.append(env.render(mode = 'rgb_array'))
    nz_obs =
    action =
    obs, reward, done, info =
    obs =
    total_reward += reward
    if done:
        break
env.close()
print('Total Reward : %.2f'%total_reward)
display_frames_as_gif(frames)
```