```
import gym
envs = gym.envs.registry.all()
total_envs = len(envs)
print(f"Total number of environments: {total_envs}")
```

```
Total number of environments: 44
/usr/local/lib/python3.11/dist-packages/gym/envs/registration.py:421: UserWarning: WARN: The `registry.all` method is deprecated. P
  logger.warn(
```

```
import gym # Retrieve all registered environments
envs = gym.envs.registry.all() # Print the names of all environments
env_names = sorted([env_spec.id for env_spec in envs])
for name in env_names:
  print(name)
```

```
Acrobot-v1
Ant-v2
Ant-v3
Ant-v4
BipedalWalker-v3
BipedalWalkerHardcore-v3
Blackjack-v1
CarRacing-v2
CartPole-v0
CartPole-v1
CliffWalking-v0
FrozenLake-v1
FrozenLake8x8-v1
HalfCheetah-v2
HalfCheetah-v3
HalfCheetah-v4
Hopper-v2
Hopper-v3
Hopper-v4
Humanoid-v2
Humanoid-v3
Humanoid-v4
HumanoidStandup-v2
HumanoidStandup-v4
InvertedDoublePendulum-v2
InvertedDoublePendulum-v4
InvertedPendulum-v2
InvertedPendulum-v4
LunarLander-v2
LunarLanderContinuous-v2
MountainCar-v0
MountainCarContinuous-v0
Pendulum-v1
Pusher-v2
Pusher-v4
Reacher-v2
Reacher-v4
Swimmer-v2
Swimmer-v3
Swimmer-v4
Taxi-v3
Walker2d-v2
Walker2d-v3
Walker2d-v4
```

```
!pip uninstall numpy
!pip install numpy==1.23.5
```

```
Found existing installation: numpy 1.23.5
Uninstalling numpy-1.23.5:
  Would remove:
    /usr/local/bin/f2py
    /usr/local/bin/f2py3
    /usr/local/bin/f2py3.11
    /usr/local/lib/python3.11/dist-packages/numpy-1.23.5.dist-info/*
    /usr/local/lib/python3.11/dist-packages/numpy.libs/libgfortran-040039e1.so.5.0.0
    /usr/local/lib/python3.11/dist-packages/numpy.libs/libopenblas64_p-r0-742d56dc.3.20.so
    /usr/local/lib/python3.11/dist-packages/numpy.libs/libquadmath-96973f99.so.0.0.0
    /usr/local/lib/python3.11/dist-packages/numpy/*
Proceed (Y/n)? y
  Successfully uninstalled numpy-1.23.5
Collecting numpy==1.23.5
  Using cached numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.3 kB)
Using cached numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
Installing collected packages: numpy
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
jaxlib 0.5.1 requires numpy>=1.25, but you have numpy 1.23.5 which is incompatible.
arviz 0.22.0 requires numpy>=1.26.0, but you have numpy 1.23.5 which is incompatible.
jax 0.5.2 requires numpy>=1.25, but you have numpy 1.23.5 which is incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.23.5 which is incompatible.
geopandas 1.1.1 requires numpy>=1.24, but you have numpy 1.23.5 which is incompatible.
scikit-image 0.25.2 requires numpy>=1.24, but you have numpy 1.23.5 which is incompatible.
pymc 5.25.1 requires numpy>=1.25.0, but you have numpy 1.23.5 which is incompatible.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.23.5 which is incompatible
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.23.5 which is incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.23.5 which is incompatible.
treescope 0.1.9 requires numpy>=1.25.2, but you have numpy 1.23.5 which is incompatible.
chex 0.1.90 requires numpy>=1.24.1, but you have numpy 1.23.5 which is incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.23.5 which is incompatible.
xarray 2025.7.1 requires numpy>=1.26, but you have numpy 1.23.5 which is incompatible.
scipy 1.16.0 requires numpy<2.6,>=1.25.2, but you have numpy 1.23.5 which is incompatible.
imbalanced-learn 0.13.0 requires numpy<3,>=1.24.3, but you have numpy 1.23.5 which is incompatible.
xarray-einstats 0.9.1 requires numpy>=1.25, but you have numpy 1.23.5 which is incompatible.
db-dtypes 1.4.3 requires numpy>=1.24.0, but you have numpy 1.23.5 which is incompatible.
bigframes 2.12.0 requires numpy>=1.24.0, but you have numpy 1.23.5 which is incompatible.
albucore 0.0.24 requires numpy>=1.24.4, but you have numpy 1.23.5 which is incompatible.
blosc2 3.6.1 requires numpy>=1.26, but you have numpy 1.23.5 which is incompatible.
albumentations 2.0.8 requires numpy>=1.24.4, but you have numpy 1.23.5 which is incompatible.
Successfully installed numpy-1.23.5
WARNING: The following packages were previously imported in this runtime:
  [numpy]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

```
import gym
import numpy as np
env = gym.make('CartPole-v1', new_step_api=True)
state = env.reset()

print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")


# State space
def describe_state(state):
    """
    This function prints out the individual components of the state
    State is a tuple (x, x_dot, theta, theta_dot)
    """
    cart_position, cart_velocity, pole_angle, pole_velocity = state
    print(f"Cart Position: {cart_position}")
    print(f"Cart Velocity: {cart_velocity}")
    print(f"Pole Angle: {pole_angle}")
    print(f"Pole Velocity: {pole_velocity}")

# Example of an initial state in CartPole
print("Initial State:")
describe_state(state)

# Action space exploration
# In CartPole, there are two actions: 0 (push left) and 1 (push right)
actions = {0: "Move Left", 1: "Move Right"}
for action in actions:
    print(f"Action {action}: {actions[action]}")

# Simulate a few steps to see state transitions and rewards
num_steps = 5
print("\nSimulating a few steps:")
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, done, info, _ = env.step(action)
```

```
        print(f"\nStep {step + 1}:")
        print(f"Action taken: {actions[action]}")
        print("Next State:")
        describe_state(next_state)
        print(f"Reward: {reward}")
        print(f"Done: {done}")

    env.close()
```

```
Cart Position: -0.02028689719736576
Cart Velocity: 0.01866157539188862
Pole Angle: -0.018840136006474495
Pole Velocity: -0.004083896055817604
Action 0: Move Left
Action 1: Move Right

Simulating a few steps:

Step 1:
Action taken: Move Left
Next State:
Cart Position: -0.01991366595029831
Cart Velocity: -0.17618519067764282
Pole Angle: -0.018921812996268272
Pole Velocity: 0.282595694065094
Reward: 1.0
Done: False

Step 2:
Action taken: Move Right
Next State:
Cart Position: -0.023437369614839554
Cart Velocity: 0.019201476126909256
Pole Angle: -0.013269899412989616
Pole Velocity: -0.01599450409412384
Reward: 1.0
Done: False

Step 3:
Action taken: Move Right
Next State:
Cart Position: -0.02305334061384201
Cart Velocity: 0.2145112007856369
Pole Angle: -0.013589790090918541
Pole Velocity: -0.3128345310688019
Reward: 1.0
Done: False

Step 4:
Action taken: Move Left
Next State:
Cart Position: -0.018763115629553795
Cart Velocity: 0.01958545297384262
Pole Angle: -0.01984648033976555
Pole Velocity: -0.024468187242746353
Reward: 1.0
Done: False

Step 5:
Action taken: Move Right
Next State:
Cart Position: -0.018371406942605972
Cart Velocity: 0.21498630940914154
Pole Angle: -0.020335843786597252
Pole Velocity: -0.32334622740745544
Reward: 1.0
Done: False
```

```
env = gym.make('FrozenLake-v1', new_step_api=True)

# Reset the environment to get the initial state
state = env.reset()

# Display the action space and observation space
print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")

# Define MDP Components for FrozenLake

# State space
def describe_state(state):
    """
    This function prints out the individual components of the state
    In FrozenLake, the state is an integer representing the agent's position.
    """
    print(f"Agent's Position: {state}")
```

```
# Example of an initial state in FrozenLake
print("Initial State:")
describe_state(state)

# Action space exploration
# In FrozenLake, there are four actions: 0 (Left), 1 (Down), 2 (Right), 3 (Up)
actions = {0: "Move Left", 1: "Move Down", 2: "Move Right", 3: "Move Up"}
for action in actions:
    print(f"Action {action}: {actions[action]}")

# Simulate a few steps to see state transitions and rewards
num_steps = 5
print("\nSimulating a few steps:")
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, done, info, _ = env.step(action)

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print("Next State:")
    describe_state(next_state)
    print(f"Reward: {reward}")
    print(f"Done: {done}")

# Close the environment when done
env.close()
```

```
Action Space: Discrete(4)
Observation Space: Discrete(16)
Initial State:
Agent's Position: 0
Action 0: Move Left
Action 1: Move Down
Action 2: Move Right
Action 3: Move Up

Simulating a few steps:

Step 1:
Action taken: Move Up
Next State:
Agent's Position: 1
Reward: 0.0
Done: False

Step 2:
Action taken: Move Right
Next State:
Agent's Position: 5
Reward: 0.0
Done: True

Step 3:
Action taken: Move Up
Next State:
Agent's Position: 5
Reward: 0
Done: True

Step 4:
Action taken: Move Down
Next State:
Agent's Position: 5
Reward: 0
Done: True

Step 5:
Action taken: Move Up
Next State:
Agent's Position: 5
Reward: 0
Done: True
```

```
env = gym.make('MountainCar-v0', new_step_api=True)

# Reset the environment to get the initial state
state = env.reset()

# Display the action space and observation space
print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")

# Define MDP Components for MountainCar

# State space
```

```
def describe_state(state):
    """
    This function prints out the individual components of the state
    State is a tuple (position, velocity)
    """
    position, velocity = state
    print(f"Car Position: {position}")
    print(f"Car Velocity: {velocity}")

# Example of an initial state in MountainCar
print("Initial State:")
describe_state(state)

# Action space exploration
# In MountainCar, there are three actions: 0 (push left), 1 (no push), 2 (push right)
actions = {0: "Push Left", 1: "No Push", 2: "Push Right"}
for action in actions:
    print(f"Action {action}: {actions[action]}")

# Simulate a few steps to see state transitions and rewards
num_steps = 5
print("\nSimulating a few steps:")
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, done, info, _ = env.step(action)

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print("Next State:")
    describe_state(next_state)
    print(f"Reward: {reward}")
    print(f"Done: {done}")

# Close the environment when done
env.close()
```

```
Action Space: Discrete(3)
Observation Space: Box([-1.2  -0.07], [0.6  0.07], (2,), float32)
Initial State:
Car Position: -0.48356401920318604
Car Velocity: 0.0
Action 0: Push Left
Action 1: No Push
Action 2: Push Right

Simulating a few steps:

Step 1:
Action taken: No Push
Next State:
Car Position: -0.48386356234550476
Car Velocity: -0.00029953938792459667
Reward: -1.0
Done: False

Step 2:
Action taken: Push Left
Next State:
Car Position: -0.48546040058135986
Car Velocity: -0.0015968482475727797
Reward: -1.0
Done: False

Step 3:
Action taken: Push Left
Next State:
Car Position: -0.48834267258644104
Car Velocity: -0.002882262459024787
Reward: -1.0
Done: False

Step 4:
Action taken: Push Right
Next State:
Car Position: -0.49048885703086853
Car Velocity: -0.0021461904980242252
Reward: -1.0
Done: False

Step 5:
Action taken: No Push
Next State:
Car Position: -0.49288296699523926
Car Velocity: -0.00239410693757236
Reward: -1.0
Done: False
```

```
env = gym.make('Blackjack-v1', new_step_api=True)

# Reset the environment to get the initial state
state = env.reset()

# Display the action space and observation space
print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")

# Define MDP Components for Blackjack

# State space
def describe_state(state):
    """
    This function prints out the individual components of the state
    State is a tuple (player_sum, dealer_card, usable_ace)
    """
    player_sum, dealer_card, usable_ace = state
    print(f"Player's Sum: {player_sum}")
    print(f"Dealer's Card: {dealer_card}")
    print(f"Usable Ace: {usable_ace}")

# Example of an initial state in Blackjack
print("Initial State:")
describe_state(state)

# Action space exploration
# In Blackjack, there are two actions: 0 (stick) and 1 (hit)
actions = {0: "Stick", 1: "Hit"}
for action in actions:
    print(f"Action {action}: {actions[action]}")

# Simulate a few steps to see state transitions and rewards
num_steps = 5
print("\nSimulating a few steps:")
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, done, info, _ = env.step(action)

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print("Next State:")
    describe_state(next_state)
    print(f"Reward: {reward}")
    print(f"Done: {done}")

# Close the environment when done
env.close()
```

```
Action Space: Discrete(2)
Observation Space: Tuple(Discrete(32), Discrete(11), Discrete(2))
Initial State:
Player's Sum: 17
Dealer's Card: 10
Usable Ace: True
Action 0: Stick
Action 1: Hit

Simulating a few steps:

Step 1:
Action taken: Stick
Next State:
Player's Sum: 17
Dealer's Card: 10
Usable Ace: True
Reward: 1.0
Done: True

Step 2:
Action taken: Stick
Next State:
Player's Sum: 17
Dealer's Card: 10
Usable Ace: True
Reward: 1.0
Done: True

Step 3:
Action taken: Hit
Next State:
Player's Sum: 15
Dealer's Card: 10
Usable Ace: False
Reward: 0.0
```

```
Done: False

Step 4:
Action taken: Hit
Next State:
Player's Sum: 24
Dealer's Card: 10
Usable Ace: False
Reward: -1.0
Done: True

Step 5:
Action taken: Hit
Next State:
Player's Sum: 27
Dealer's Card: 10
Usable Ace: False
Reward: -1.0
Done: True
```

```
env = gym.make('Taxi-v3', new_step_api=True)

# Reset the environment to get the initial state
state = env.reset()

# Display the action space and observation space
print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")

# Define MDP Components for Taxi

# State space
def describe_state(state):
    """
    This function decodes the state into its individual components.
    The state is an integer from 0 to 499.
    """
    print(f"State: {state}")

# Example of an initial state in Taxi
print("Initial State:")
describe_state(state)

# Action space exploration
# In Taxi, there are six actions: 0 (South), 1 (North), 2 (East), 3 (West), 4 (Pickup), 5 (Dropoff)
actions = {0: "South", 1: "North", 2: "East", 3: "West", 4: "Pickup", 5: "Dropoff"}
for action in actions:
    print(f"Action {action}: {actions[action]}")

# Simulate a few steps to see state transitions and rewards
num_steps = 5
print("\nSimulating a few steps:")
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, done, info, _ = env.step(action)

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print("Next State:")
    describe_state(next_state)
    print(f"Reward: {reward}")
    print(f"Done: {done}")

# Close the environment when done
env.close()
```

```
Action Space: Discrete(6)
Observation Space: Discrete(500)
Initial State:
State: 107
Action 0: South
Action 1: North
Action 2: East
Action 3: West
Action 4: Pickup
Action 5: Dropoff

Simulating a few steps:

Step 1:
Action taken: Dropoff
Next State:
State: 107
Reward: -10
Done: False
```

```
Step 2:
Action taken: East
Next State:
State: 127
Reward: -1
Done: False

Step 3:
Action taken: Pickup
Next State:
State: 127
Reward: -10
Done: False

Step 4:
Action taken: Pickup
Next State:
State: 127
Reward: -10
Done: False

Step 5:
Action taken: Dropoff
Next State:
State: 127
Reward: -10
Done: False
```

```
env = gym.make('CliffWalking-v0', new_step_api=True)

# Reset the environment to get the initial state
state = env.reset()

# Display the action space and observation space
print(f"Action Space: {env.action_space}")
print(f"Observation Space: {env.observation_space}")

# Define MDP Components for CliffWalking

# State space
def describe_state(state):
    """
    This function prints out the individual components of the state
    In CliffWalking, the state is an integer representing the agent's position.
    """
    print(f"Agent's Position: {state}")

# Example of an initial state in CliffWalking
print("Initial State:")
describe_state(state)

# Action space exploration
# In CliffWalking, there are four actions: 0 (Up), 1 (Right), 2 (Down), 3 (Left)
actions = {0: "Up", 1: "Right", 2: "Down", 3: "Left"}
for action in actions:
    print(f"Action {action}: {actions[action]}")

# Simulate a few steps to see state transitions and rewards
num_steps = 5
print("\nSimulating a few steps:")
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, done, info, _ = env.step(action)

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print("Next State:")
    describe_state(next_state)
    print(f"Reward: {reward}")
    print(f"Done: {done}")

# Close the environment when done
env.close()
```

```
Action Space: Discrete(4)
Observation Space: Discrete(48)
Initial State:
Agent's Position: 36
Action 0: Up
Action 1: Right
Action 2: Down
Action 3: Left

Simulating a few steps:
```

```
Step 1:
Action taken: Down
Next State:
Agent's Position: 36
Reward: -1
Done: False

Step 2:
Action taken: Up
Next State:
Agent's Position: 24
Reward: -1
Done: False

Step 3:
Action taken: Down
Next State:
Agent's Position: 36
Reward: -1
Done: False

Step 4:
Action taken: Left
Next State:
Agent's Position: 36
Reward: -1
Done: False

Step 5:
Action taken: Left
Next State:
Agent's Position: 36
Reward: -1
Done: False
```

Start coding or generate with AI.