```
import gym # Retrieve all registered environments
envs = gym.envs.registry.all() # Count the total number of
total_envs = len(envs)
print(f"Total number of environments: {total_envs}")
```

```
Total number of environments: 44
```

```
import gymnasium  # Retrieve all registered environments
envs = gymnasium.envs.registry # Count the total number of
total_envs = len(envs)
print(f"Total number of environments: {total_envs}")
```

```
Total number of environments: 63
```

```
import gym # Retrieve all registered environments
envs = gym.envs.registry.all() # Print the names of all
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
/usr/local/lib/python3.11/dist-packages/gym/envs/registration.py:421: UserWarning: WARN: The `registry.all` method is deprecated. P
  logger.warn(
```

```
# CartPole MDP Exploration in Jupyter Notebook

# Import necessary libraries
import gymnasium as gym # Use gymnasium as the maintained successor to gym
import numpy as np

# Load the CartPole environment
env = gym.make('CartPole-v1')

# Reset the environment to get the initial state
# gymnasium's reset returns a tuple (observation, info)
state, info = env.reset(seed=42) # Added seed for reproducibility

# 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 CartPole

# 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
    # gymnasium's step returns (observation, reward, terminated, truncated, info)
    next_state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated # done is true if either terminated or truncated is true

    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}") # Use the combined done flag

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

```
Cart Velocity: -0.006112155970185995
Pole Angle: 0.03585979342460632
Pole Velocity: 0.019736802205443382
Action 0: Move Left
Action 1: Move Right

Simulating a few steps:

Step 1:
Action taken: Move Right
Next State:
Cart Position: 0.02727336250245571
Cart Velocity: 0.18847766518592834
Pole Angle: 0.036254528909921646
Pole Velocity: -0.26141977310180664
Reward: 1.0
Done: False

Step 2:
Action taken: Move Right
Next State:
Cart Position: 0.0310429148375988
Cart Velocity: 0.3830638527870178
Pole Angle: 0.03102613240480423
Pole Velocity: -0.5424507260322571
Reward: 1.0
Done: False

Step 3:
Action taken: Move Right
Next State:
Cart Position: 0.03870419040322304
Cart Velocity: 0.5777363181114197
Pole Angle: 0.020177118480205536
Pole Velocity: -0.8251987099647522
Reward: 1.0
```

```
Next State:
Cart Position: 0.05025891959667206
Cart Velocity: 0.3823443055152893
Pole Angle: 0.0036731448490172625
Pole Velocity: -0.5262386202812195
Reward: 1.0
Done: False

Step 5:
Action taken: Move Right
Next State:
Cart Position: 0.05790580436587334
Cart Velocity: 0.5774143934249878
Pole Angle: -0.006851627957075834
Pole Velocity: -0.8177618980407715
Reward: 1.0
Done: False
```

```
# Load the FrozenLake environment
# You can choose between 'FrozenLake-v1' (slippery) and 'FrozenLake8x8-v1' (slippery 8x8)
# or 'FrozenLake-v1' with is_slippery=False
env_name = 'FrozenLake-v1'
env = gym.make(env_name)

# Reset the environment to get the initial state
state, info = env.reset(seed=42)

# 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: The states in FrozenLake are the grid cells, represented by integers from 0 to N-1, where N is the total number of cells.
print("\nState Space:")
print(f"Number of states: {env.observation_space.n}")
print("States represent grid locations.")
# Example: For a 4x4 grid, states are 0 to 15.

# Action space: The actions are moving in four directions: Left (0), Down (1), Right (2), Up (3).
print("\nAction Space Exploration:")
actions = {0: "Left", 1: "Down", 2: "Right", 3: "Up"}
for action in actions:
    print(f"Action {action}: {actions[action]}")


# Transition probability and Reward
print("\nExploring Transitions and Rewards (for a few steps):")
num_steps = 5
env.reset(seed=42) # Reset to a known state for demonstration
for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print(f"Current State: {state}") # Print the state *before* the action
    print(f"Next State: {next_state}")
    print(f"Reward: {reward}")
    print(f"Done: {done}")

    state = next_state # Update state for the next step

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

```
Action Space: Discrete(4)
Observation Space: Discrete(16)

State Space:
Number of states: 16
States represent grid locations.

Action Space Exploration:
Action 0: Left
Action 1: Down
Action 2: Right
Action 3: Up

Exploring Transitions and Rewards (for a few steps):

Step 1:
Action taken: Up
```

```
Current State: 0
Next State: 0
Reward: 0.0
Done: False

Step 2:
Action taken: Down
Current State: 0
Next State: 1
Reward: 0.0
Done: False

Step 3:
Action taken: Up
Current State: 1
Next State: 0
Reward: 0.0
Done: False

Step 4:
Action taken: Up
Current State: 0
Next State: 1
Reward: 0.0
Done: False

Step 5:
Action taken: Right
Current State: 1
Next State: 1
Reward: 0.0
Done: False
```

```
import gymnasium as gym
import numpy as np

# Load the MountainCar environment
env = gym.make('MountainCar-v0')

# Reset the environment
state, info = env.reset(seed=42)

# Display 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: The state is a 2D continuous space: [position, velocity]
print("\nState Space:")
print(f"Observation Space Low: {env.observation_space.low}")
print(f"Observation Space High: {env.observation_space.high}")
print("State represents [position, velocity].")

# Action space: The actions are discrete: 0 (push left), 1 (do nothing), 2 (push right)
print("\nAction Space Exploration:")
actions = {0: "Push Left", 1: "Do Nothing", 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
env.reset(seed=42) # Reset to a known state for demonstration
print("\nExploring Transitions and Rewards (for a few steps):")
state, info = env.reset(seed=42) # Get the initial state after reset

for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print(f"Current State: {state}") # Print the state *before* the action
    print(f"Next State: {next_state}")
    print(f"Reward: {reward}")
    print(f"Done: {done}")

    state = next_state # Update state for the next step

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

```
Action Space: Discrete(3)
Observation Space: Box([-1.2  -0.07], [0.6  0.07], (2,), float32)

State Space:
Observation Space Low: [-1.2  -0.07]
Observation Space High: [0.6  0.07]
State represents [position, velocity].

Action Space Exploration:
Action 0: Push Left
Action 1: Do Nothing
Action 2: Push Right

Exploring Transitions and Rewards (for a few steps):

Step 1:
Action taken: Push Right
Current State: [-0.4452088  0.       ]
Next State: [-4.4479132e-01  4.1747934e-04]
Reward: -1.0
Done: False

Step 2:
Action taken: Push Right
Current State: [-4.4479132e-01  4.1747934e-04]
Next State: [-0.4439594   0.00083191]
Reward: -1.0
Done: False

Step 3:
Action taken: Push Right
Current State: [-0.4439594   0.00083191]
Next State: [-0.4427191   0.00124029]
Reward: -1.0
Done: False

Step 4:
Action taken: Do Nothing
Current State: [-0.4427191   0.00124029]
Next State: [-0.44207948  0.00063962]
Reward: -1.0
Done: False

Step 5:
Action taken: Push Right
Current State: [-0.44207948  0.00063962]
Next State: [-0.4410452  0.0010343]
Reward: -1.0
Done: False
```

```python
import gymnasium as gym
import numpy as np

# Load the Blackjack environment
# The Blackjack environment in Gymnasium is 'Blackjack-v1'
env = gym.make('Blackjack-v1')

# Reset the environment
# The state in Blackjack is a tuple: (player's current sum, dealer's showing card, whether the player has a usable ace)
state, info = env.reset(seed=42)

# Display 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: The state is a tuple (player's current sum, dealer's showing card, whether the player has a usable ace)
print("\nState Space:")
print(f"Initial State: {state}")
print("State represents (player's current sum, dealer's showing card, whether the player has a usable ace).")
print("  - Player's current sum: Integer from 4 to 21.")
print("  - Dealer's showing card: Integer from 1 (Ace) to 10 (10 or face card).")
print("  - Usable ace: Boolean (True if the player has an ace they can count as 11 without busting, False otherwise).")


# Action space: The actions are discrete: 0 (stick), 1 (hit)
print("\nAction Space Exploration:")
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
# In Blackjack, a 'step' corresponds to the player taking an action (hit or stick)
# The episode terminates when the player sticks, busts, or gets 21.
```

```
num_steps_to_simulate = 3 # Simulate up to 3 player actions

print("\nExploring Transitions and Rewards (simulating player actions):")
# Reset to a known state for demonstration, though the initial state is random in Blackjack
state, info = env.reset(seed=42)
print(f"Initial State: {state}")

for step in range(num_steps_to_simulate):
    # In a real scenario, an agent would choose an action based on the state.
    # Here, we'll just take a random action (hit or stick).
    # Taking 'stick' (action 0) will usually end the episode quickly.
    # Let's try taking 'hit' (action 1) for a few steps if possible.
    action = 1 # Try to hit

    # Check if the action is valid in the current state (always true for hit/stick in Blackjack before episode ends)
    if action in [0, 1]:
        print(f"\nStep {step + 1}:")
        print(f"Action taken: {actions[action]}")
        print(f"Current State: {state}") # Print the state *before* the action

        next_state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated

        print(f"Next State: {next_state}")
        print(f"Reward: {reward}")
        print(f"Done: {done}")

        state = next_state # Update state for the next step

        if done:
            print("Episode finished.")
            break # Stop simulating if the episode is done
    else:
        print(f"Invalid action {action} taken.")
        break

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

```
Action Space: Discrete(2)
Observation Space: Tuple(Discrete(32), Discrete(11), Discrete(2))

State Space:
Initial State: (15, 2, 0)
State represents (player's current sum, dealer's showing card, whether the player has a usable ace).
  - Player's current sum: Integer from 4 to 21.
  - Dealer's showing card: Integer from 1 (Ace) to 10 (10 or face card).
  - Usable ace: Boolean (True if the player has an ace they can count as 11 without busting, False otherwise).

Action Space Exploration:
Action 0: Stick
Action 1: Hit

Exploring Transitions and Rewards (simulating player actions):
Initial State: (15, 2, 0)

Step 1:
Action taken: Hit
Current State: (15, 2, 0)
Next State: (25, 2, 0)
Reward: -1.0
Done: True
Episode finished.
```

```
import gymnasium as gym
import numpy as np

# Load the Taxi environment
# The Taxi environment in Gymnasium is 'Taxi-v3'
env = gym.make('Taxi-v3')

# Reset the environment
# The state in Taxi is a single integer representing the taxi's location,
# passenger's location, and destination location.
state, info = env.reset(seed=42)

# Display 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: The state is a single integer representing a combination of
# taxi location, passenger location, and destination.
```

```
print("\nState Space:")
print(f"Initial State: {state}")
print(f"Number of states: {env.observation_space.n}")
print("States encode the taxi's position, passenger's current location, and passenger's destination.")

# Action space: The actions are discrete: 0 (South), 1 (North), 2 (East),
# 3 (West), 4 (Pickup), 5 (Dropoff).
print("\nAction Space Exploration:")
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("\nExploring Transitions and Rewards (for a few steps):")
state, info = env.reset(seed=42) # Get the initial state after reset
print(f"Initial State: {state}")

for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print(f"Current State: {state}") # Print the state *before* the action
    print(f"Next State: {next_state}")
    print(f"Reward: {reward}")
    print(f"Done: {done}")

    state = next_state # Update state for the next step

    if done:
        print("Episode finished.")
        break # Stop simulating if the episode is done

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

```
Action Space: Discrete(6)
Observation Space: Discrete(500)

State Space:
Initial State: 386
Number of states: 500
States encode the taxi's position, passenger's current location, and passenger's destination.

Action Space Exploration:
Action 0: South
Action 1: North
Action 2: East
Action 3: West
Action 4: Pickup
Action 5: Dropoff

Exploring Transitions and Rewards (for a few steps):
Initial State: 386

Step 1:
Action taken: North
Current State: 386
Next State: 286
Reward: -1
Done: False

Step 2:
Action taken: West
Current State: 286
Next State: 266
Reward: -1
Done: False

Step 3:
Action taken: Pickup
Current State: 266
Next State: 266
Reward: -10
Done: False

Step 4:
Action taken: Pickup
Current State: 266
Next State: 266
Reward: -10
Done: False
```

```
Step 5:
Action taken: Dropoff
Current State: 266
Next State: 266
Reward: -10
Done: False
```

```
import gymnasium as gym
import numpy as np

# Load the CliffWalking environment
# The CliffWalking environment in Gymnasium is 'CliffWalking-v1'
env = gym.make('CliffWalking-v1')

# Reset the environment
# The state in CliffWalking is a single integer representing the agent's position on the grid.
state, info = env.reset(seed=42)

# Display 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: The state is a single integer representing the agent's position on the grid.
print("\nState Space:")
print(f"Initial State: {state}")
print(f"Number of states: {env.observation_space.n}")
print("States represent the agent's position on the grid.")

# Action space: The actions are discrete: 0 (Up), 1 (Right), 2 (Down), 3 (Left).
print("\nAction Space Exploration:")
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("\nExploring Transitions and Rewards (for a few steps):")
state, info = env.reset(seed=42) # Get the initial state after reset
print(f"Initial State: {state}")

for step in range(num_steps):
    action = env.action_space.sample()  # Random action
    next_state, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated

    print(f"\nStep {step + 1}:")
    print(f"Action taken: {actions[action]}")
    print(f"Current State: {state}") # Print the state *before* the action
    print(f"Next State: {next_state}")
    print(f"Reward: {reward}")
    print(f"Done: {done}")

    state = next_state # Update state for the next step

    if done:
        print("Episode finished.")
        break # Stop simulating if the episode is done

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

```
Action Space: Discrete(4)
Observation Space: Discrete(48)

State Space:
Initial State: 36
Number of states: 48
States represent the agent's position on the grid.

Action Space Exploration:
Action 0: Up
Action 1: Right
Action 2: Down
Action 3: Left

Exploring Transitions and Rewards (for a few steps):
Initial State: 36

Step 1:
Action taken: Right
Current State: 36
Next State: 36
```

```
Reward: -100
Done: False

Step 2:
Action taken: Right
Current State: 36
Next State: 36
Reward: -100
Done: False

Step 3:
Action taken: Up
Current State: 36
Next State: 24
Reward: -1
Done: False

Step 4:
Action taken: Left
Current State: 24
Next State: 24
Reward: -1
Done: False

Step 5:
Action taken: Right
Current State: 24
Next State: 25
Reward: -1
Done: False
```