States

The student has three primary locations and two conditions (attending class or eating):

- **States**:
  - `S1`: Hostel, attending class
  - `S2`: Hostel, eating food
  - `S3`: Academic Building, attending class
  - `S4`: Academic Building, eating food
  - `S5`: Canteen, attending class
  - `S6`: Canteen, eating food

Actions

- **Actions**:
  - `A1`: Go to Hostel
  - `A2`: Go to Academic Building
  - `A3`: Go to Canteen
  - `A4`: Stay at the current location

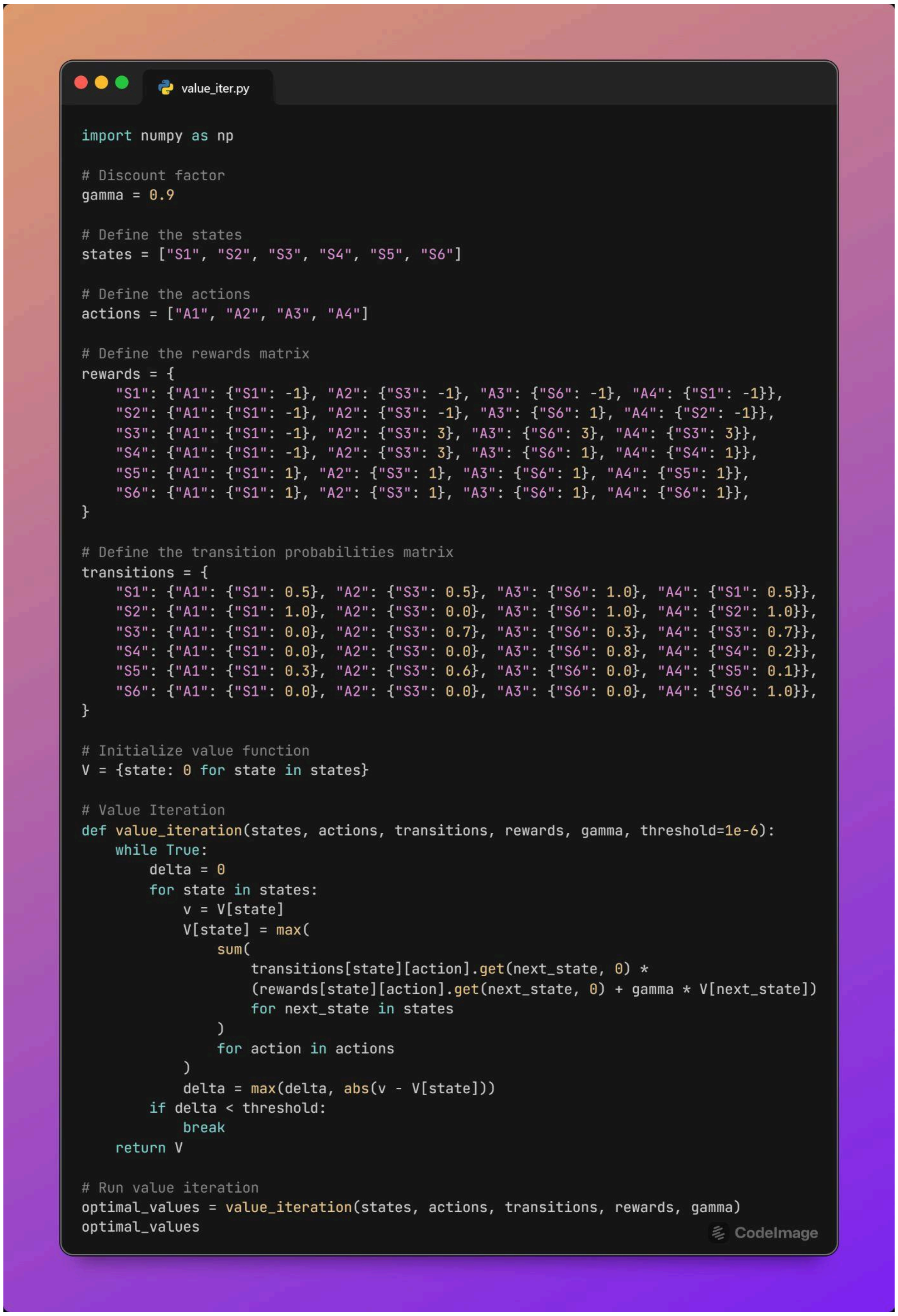
Transition Probabilities and Rewards

From State	Action	To State	Probability	Reward
S1 (Hostel, Class)	A2 (Go to Academic Building)	S3 (Academic Building, Class)	0.5	-1
S1 (Hostel, Class)	A1 (Stay in Hostel)	S1 (Hostel, Class)	0.5	-1
S1 (Hostel, Class)	A3 (Go to Canteen)	S6 (Canteen, Eating)	1.0	-1
S2 (Hostel, Eating)	A3 (Go to Canteen)	S6 (Canteen, Eating)	1.0	1
S3 (Academic Building, Class)	A4 (Stay in Academic Building)	S3 (Academic Building, Class)	0.7	3
S3 (Academic Building, Class)	A3 (Go to Canteen)	S6 (Canteen, Eating)	0.3	3
S4 (Academic Building, Eating)	A3 (Go to Canteen)	S6 (Canteen, Eating)	0.8	1

S4 (Academic Building, Eating)	A4 (Stay in Academic Building)	S4 (Academic Building, Eating)	0.2	1
S5 (Canteen, Class)	A2 (Go to Academic Building)	S3 (Academic Building, Class)	0.6	1
S5 (Canteen, Class)	A1 (Go to Hostel)	S1 (Hostel, Class)	0.3	1
S5 (Canteen, Class)	A4 (Stay in Canteen)	S5 (Canteen, Class)	0.1	1
S6 (Canteen, Eating)	A4 (Stay in Canteen)	S6 (Canteen, Eating)	1.0	1

Step 2: Value Iteration

Code:


value_iter.py

```python
import numpy as np

# Discount factor
gamma = 0.9

# Define the states
states = ["S1", "S2", "S3", "S4", "S5", "S6"]

# Define the actions
actions = ["A1", "A2", "A3", "A4"]

# Define the rewards matrix
rewards = {
    "S1": {"A1": {"S1": -1}, "A2": {"S3": -1}, "A3": {"S6": -1}, "A4": {"S1": -1}},
    "S2": {"A1": {"S1": -1}, "A2": {"S3": -1}, "A3": {"S6": 1}, "A4": {"S2": -1}},
    "S3": {"A1": {"S1": -1}, "A2": {"S3": 3}, "A3": {"S6": 3}, "A4": {"S3": 3}},
    "S4": {"A1": {"S1": -1}, "A2": {"S3": 3}, "A3": {"S6": 1}, "A4": {"S4": 1}},
    "S5": {"A1": {"S1": 1}, "A2": {"S3": 1}, "A3": {"S6": 1}, "A4": {"S5": 1}},
    "S6": {"A1": {"S1": 1}, "A2": {"S3": 1}, "A3": {"S6": 1}, "A4": {"S6": 1}},
}

# Define the transition probabilities matrix
transitions = {
    "S1": {"A1": {"S1": 0.5}, "A2": {"S3": 0.5}, "A3": {"S6": 1.0}, "A4": {"S1": 0.5}},
    "S2": {"A1": {"S1": 1.0}, "A2": {"S3": 0.0}, "A3": {"S6": 1.0}, "A4": {"S2": 1.0}},
    "S3": {"A1": {"S1": 0.0}, "A2": {"S3": 0.7}, "A3": {"S6": 0.3}, "A4": {"S3": 0.7}},
    "S4": {"A1": {"S1": 0.0}, "A2": {"S3": 0.0}, "A3": {"S6": 0.8}, "A4": {"S4": 0.2}},
    "S5": {"A1": {"S1": 0.3}, "A2": {"S3": 0.6}, "A3": {"S6": 0.0}, "A4": {"S5": 0.1}},
    "S6": {"A1": {"S1": 0.0}, "A2": {"S3": 0.0}, "A3": {"S6": 0.0}, "A4": {"S6": 1.0}},
}

# Initialize value function
V = {state: 0 for state in states}

# Value Iteration
def value_iteration(states, actions, transitions, rewards, gamma, threshold=1e-6):
    while True:
        delta = 0
        for state in states:
            v = V[state]
            V[state] = max(
                sum(
                    transitions[state][action].get(next_state, 0) *
                    (rewards[state][action].get(next_state, 0) + gamma * V[next_state])
                    for next_state in states
                )
                for action in actions
            )
            delta = max(delta, abs(v - V[state]))
        if delta < threshold:
            break
    return V

# Run value iteration
optimal_values = value_iteration(states, actions, transitions, rewards, gamma)
optimal_values
```

Results:
Values obtained from Value Iteration
S1 (Hostel, Class): 8.00
S2 (Hostel, Eating): 10.00
S3 (Academic Building, Class): 5.68
S4 (Academic Building, Eating): 8.00
S5 (Canteen, Class): 3.66
S6 (Canteen, Eating): 10.00

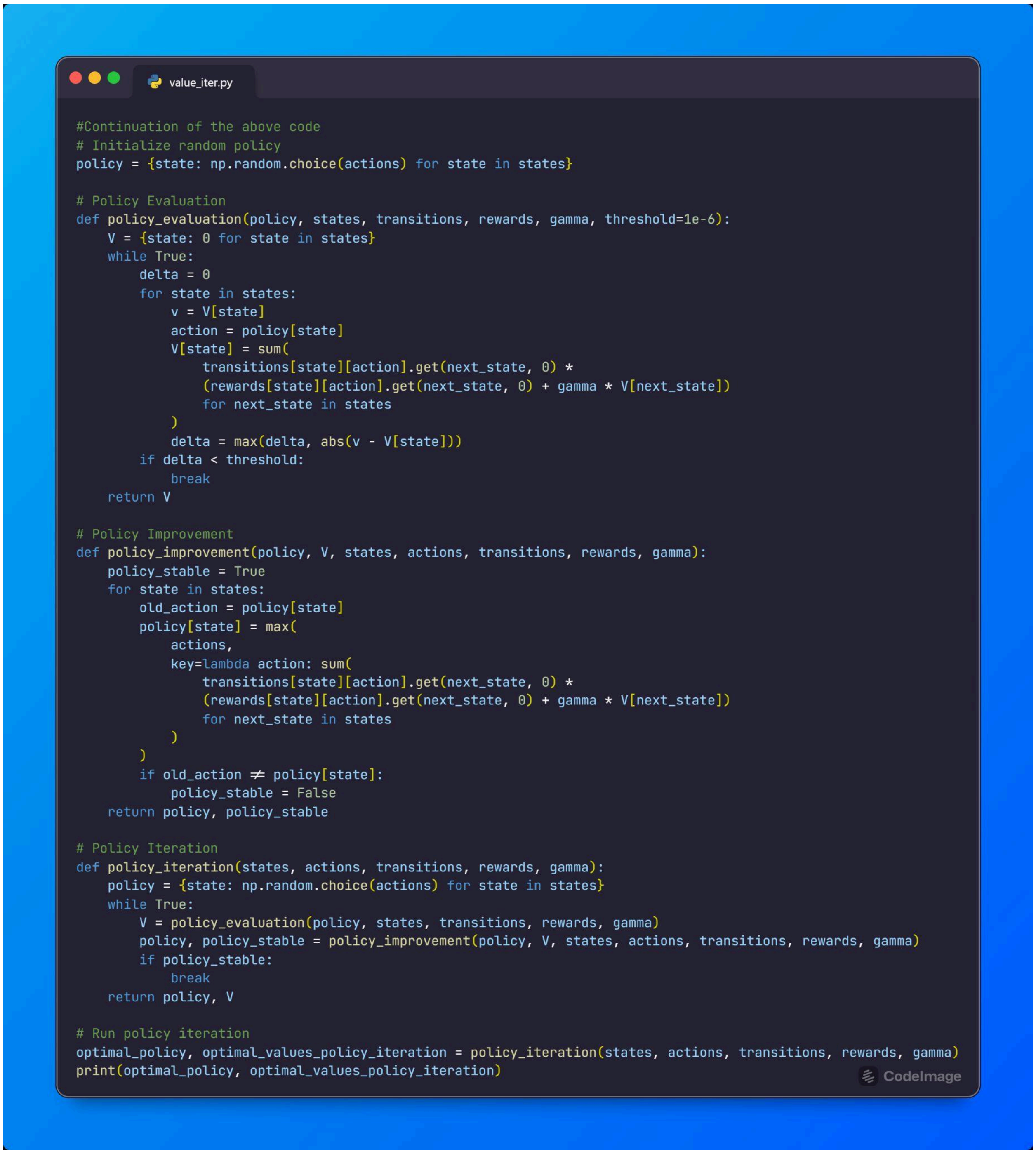
Optimal Policy:

- **S1 (Hostel, Class)**: `A3` (Go to Canteen)
- **S2 (Hostel, Eating)**: `A3` (Go to Canteen)
- **S3 (Academic Building, Class)**: `A2` (Stay in Academic Building)
- **S4 (Academic Building, Eating)**: `A3` (Go to Canteen)
- **S5 (Canteen, Class)**: `A2` (Go to Academic Building)
- **S6 (Canteen, Eating)**: `A4` (Stay in Canteen)

Optimal Values from Policy Iteration:

The values are the same as those obtained from value iteration:

- **S1 (Hostel, Class)**: 8.00
- **S2 (Hostel, Eating)**: 10.00
- **S3 (Academic Building, Class)**: 5.68
- **S4 (Academic Building, Eating)**: 8.00
- **S5 (Canteen, Class)**: 3.66
- **S6 (Canteen, Eating)**: 10.00



```python
#Continuation of the above code
# Initialize random policy
policy = {state: np.random.choice(actions) for state in states}

# Policy Evaluation
def policy_evaluation(policy, states, transitions, rewards, gamma, threshold=1e-6):
    V = {state: 0 for state in states}
    while True:
        delta = 0
        for state in states:
            v = V[state]
            action = policy[state]
            V[state] = sum(
                transitions[state][action].get(next_state, 0) *
                (rewards[state][action].get(next_state, 0) + gamma * V[next_state])
                for next_state in states
            )
            delta = max(delta, abs(v - V[state]))
        if delta < threshold:
            break
    return V

# Policy Improvement
def policy_improvement(policy, V, states, actions, transitions, rewards, gamma):
    policy_stable = True
    for state in states:
        old_action = policy[state]
        policy[state] = max(
            actions,
            key=lambda action: sum(
                transitions[state][action].get(next_state, 0) *
                (rewards[state][action].get(next_state, 0) + gamma * V[next_state])
                for next_state in states
            )
        )
        if old_action ≠ policy[state]:
            policy_stable = False
    return policy, policy_stable

# Policy Iteration
def policy_iteration(states, actions, transitions, rewards, gamma):
    policy = {state: np.random.choice(actions) for state in states}
    while True:
        V = policy_evaluation(policy, states, transitions, rewards, gamma)
        policy, policy_stable = policy_improvement(policy, V, states, actions, transitions, rewards, gamma)
        if policy_stable:
            break
    return policy, V

# Run policy iteration
optimal_policy, optimal_values_policy_iteration = policy_iteration(states, actions, transitions, rewards, gamma)
print(optimal_policy, optimal_values_policy_iteration)
```

Value Iteration and **Policy Iteration** yielded the same optimal values for each state.

The optimal policy suggests that the student prefers to go to the canteen when in the hostel or academic building, reflecting the reward structure and transition probabilities.

Policy Iteration was able to directly determine the optimal actions for each state, while **Value Iteration** focuses on refining the value function until convergence.