

# Introduction to Reinforcement Learning

and its applications to networking

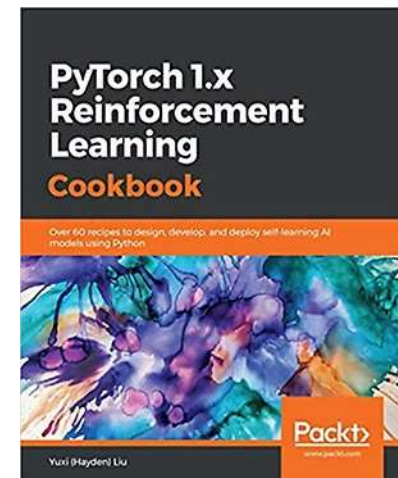
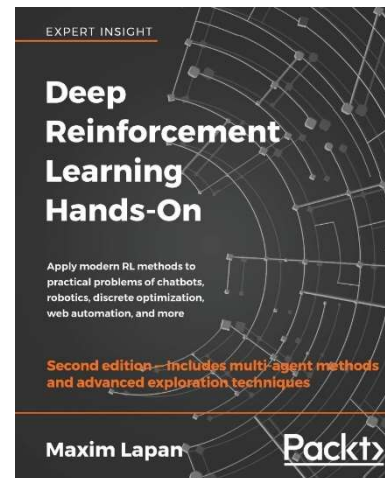
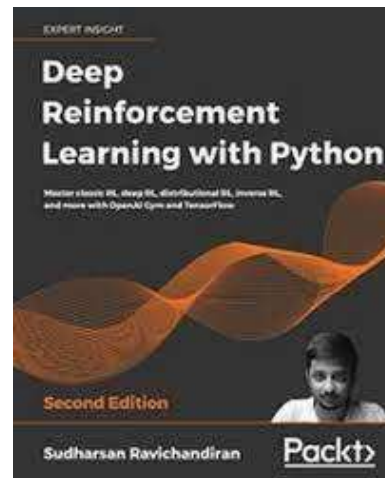
---

Dept. of Computer Engineering,  
Sogang University

This material contains contents from other sources. Do not post it on the Internet.

# Course Info

- Instructor: Jungmin So (AS1013, jso1@sogang.ac.kr)
- Lecture slides and other information will appear at <http://cyber.sogang.ac.kr>.
- Slides are based on the following books
  - Sutton and Barto, "Reinforcement Learning: An Introduction, 2nd Ed."
  - S. Ravichandiran, "Deep Reinforcement Learning with Python, 2nd Ed."
  - M. Lapan, "Deep Reinforcement Learning Hands-On, 2nd Ed."
  - Y. Liu, "PyTorch 1.x Reinforcement Learning Cookbook"



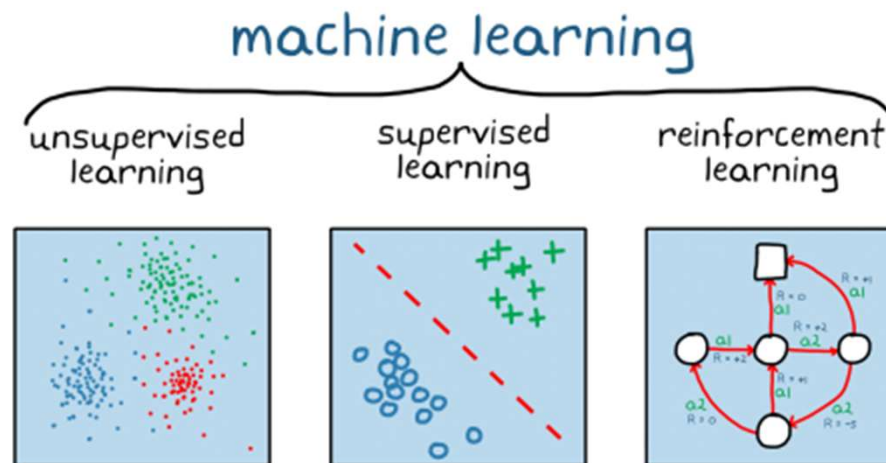
# Evaluation

- 1 Mid-term Exam
- 1 Final Exam
- 1 Individual Project
  - Using reinforcement learning to solve problems



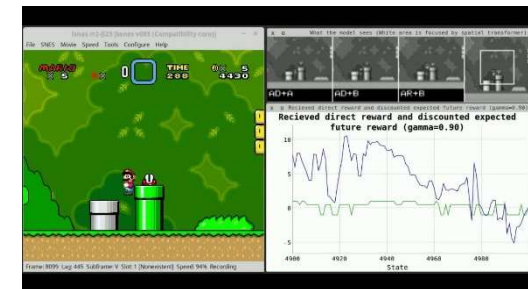
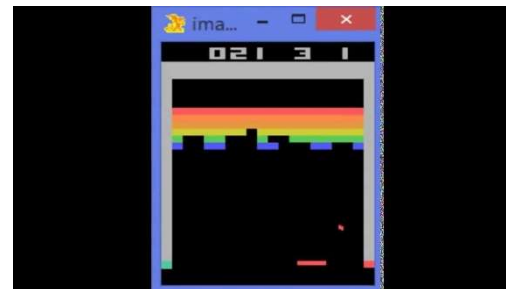
# Reinforcement Learning

- Definition from Wikipedia
  - Reinforcement learning (RL) is an area of machine learning concerned with how intelligent **agents** ought to take **actions** in an **environment** in order to maximize the notion of cumulative **reward**.
- Different types of machine learning



# Reinforcement Learning - Applications

- self-driving cars
- industry automation
- trading and finance
- natural language processing
- healthcare
- news recommendation
- gaming
- marketing and advertising
- robotics
- and many more!



# Reinforcement Learning - Topics

- introduction and concepts
- Bellman equation and dynamic programming (DP)
- Monte Carlo methods (MC)
- temporal difference (TD) learning
- deep reinforcement learning (DQN)
- policy gradient method (PG)
- actor-critic methods (AC)
- multi-agent reinforcement learning
- advanced topics
  - deep deterministic policy gradient (DDPG)
  - soft actor-critic (SAC)
  - trust region policy optimization (TRPO)
  - proximal policy optimization (PPO)

# Our goal in this course



- Understands concepts of RL and can run toy examples like FrozenLake



- Understands various methods of reinforcement learning and the differences between them



- Can apply reinforcement learning to solve new problems



- Understands math behind algorithms and why they are designed like that



- Can design new types of reinforcement learning algorithms and methods

# Fundamentals of Reinforcement Learning

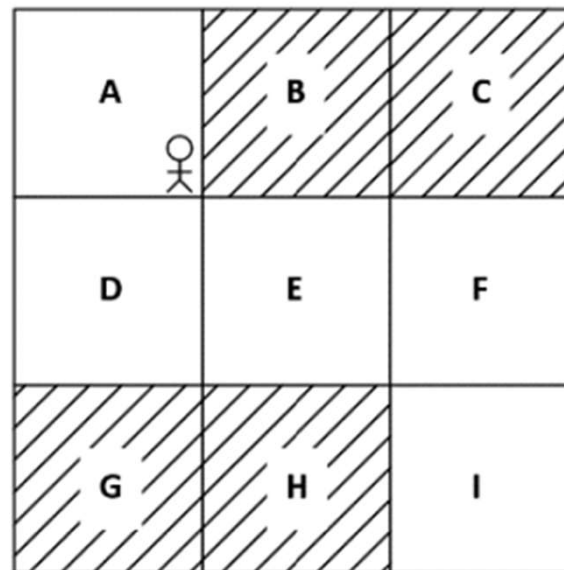
---

## Introduction



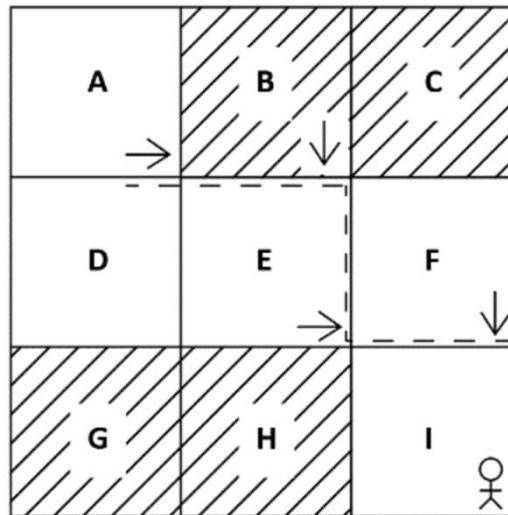
# An Example Problem

- Suppose there is a person in a grid world.
- The person starts at position A.
- His/her goal is to move to position I.
- The person can move in only two directions: right or down.
- The shaded locations (B, C, G, and H) are to be avoided, otherwise the person gets a penalty.
- What path should the person take in order to reach the goal?



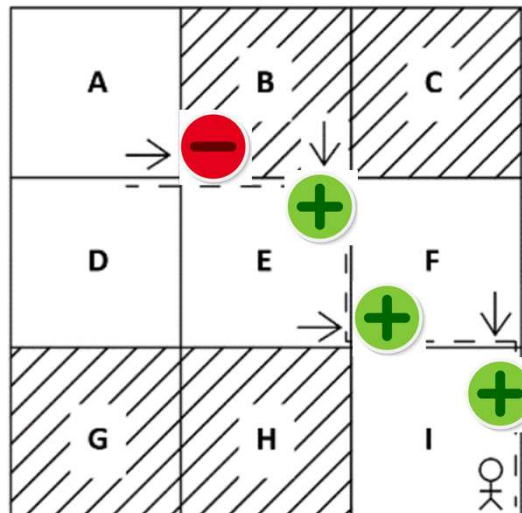
# Reinforcement Learning Approach

- Try and learn from experience!
- Iteration (**episode**) 1
  - The person (**agent**) has no experience, so it chooses to perform a random **action** among possible actions. It moves right and reaches location (**state**) B.
  - Still taking random actions, the agent moves down from B to reach state E, moves right to reach state F, and moves down to reach state I.
  - State I is defined as a **terminal state**, so the iteration ends there.



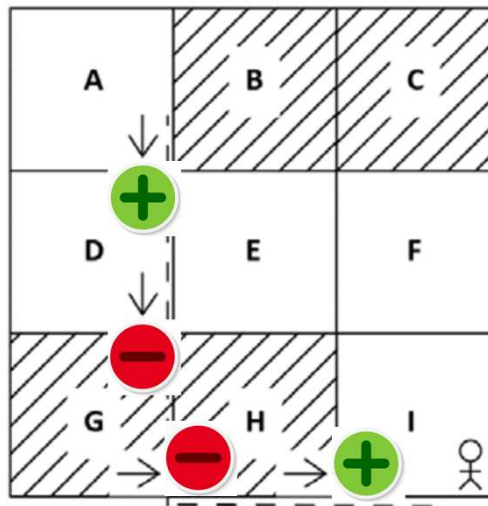
# Reinforcement Learning Approach

- How does the agent learn from experience?
  - Based on the concept of **reward**!
- Reward in the grid world
  - If the agent moves to a shaded state, it receives a negative reward.
    - The agent learns that the move was a 'bad' action.
  - If the agent moves to an unshaded state, it receives a positive reward.
    - The agent learns that the move was a 'good' action.
  - The reward is given to a specific action on a specific state.
    - (state, action) pair



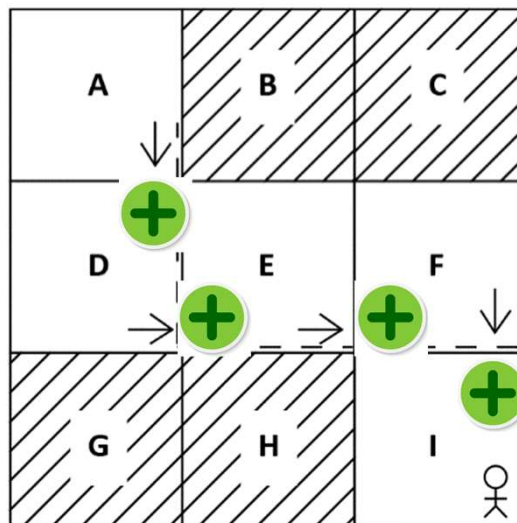
# Reinforcement Learning Approach

- Episode 2
  - This time, the agent moves down from state A to reach state D.
  - Since state D is a unshaded state, the agent receives a positive reward.
  - From this experience, the agent learns that moving down from state A is a good action.
  - Similarly, the agent moves in the path  $A \rightarrow D \rightarrow G \rightarrow H \rightarrow I$ . It receives positive rewards for actions  $A \rightarrow D$  and  $H \rightarrow I$ , but receives negative rewards for actions  $D \rightarrow G$  and  $G \rightarrow H$ .



# Reinforcement Learning Approach

- Episode 3
  - In this iteration, the agent moves in the path  $A \rightarrow D \rightarrow E \rightarrow F \rightarrow I$ . For all actions, the agent receives positive rewards.



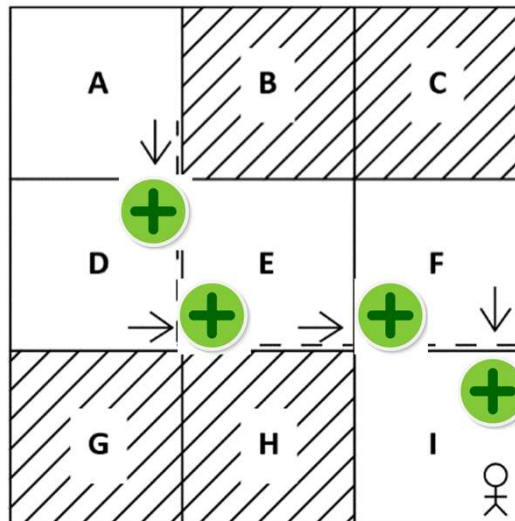
# Reinforcement Learning Approach

- Learning from experience
  - After each iteration, the agent records the reward for each (state, action) pair.



states	actions	rewards
A	move right	-1
	move down	+1

- As the experience builds up, the agent leans towards taking actions with high reward → does the task better. That's why it is called **reinforcement learning**!

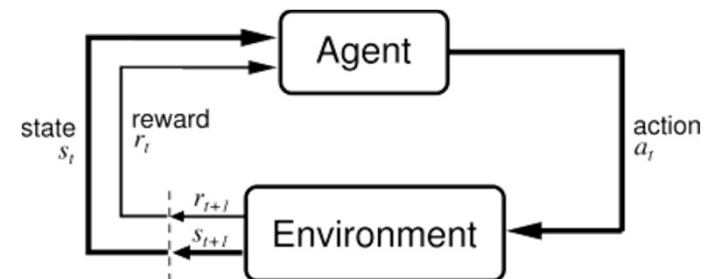


# Reinforcement Learning Approach

- Basic reinforcement learning algorithm
  - 1) The agent interacts with the environment by performing an action.
  - 2) By performing an action, the agent moves from one state to another.
  - 3) The agent receives a reward based on the action it performed.
  - 4) Based on the reward, the agent understands whether the action is good or bad.
  - 5) If the action was good, that is, if the agent received a positive reward, the agent will prefer performing that action, else the agent will try performing other actions in search of a positive reward.

# Terms used in RL

- Agent
  - a software program that learns to make intelligent decisions
    - e.g.) chess player, Mario in a Super Mario Bros. game.
- Environment
  - the world where the agent stays within
    - e.g.) chess board, the world of Mario
- State
  - a snapshot of moment in the environment (including the agent.)
    - e.g.) In a 3x3 grid world, the player can be in 1 of the 9 locations. Each location of the player is a state.
- Action
  - a move performed by the agent.
    - e.g.) Moving to a new location in the grid world, moving a chess piece in a chess board
- Reward
  - a feedback given for an action
    - e.g.) taking opponent's chess piece, reaching a goal





# Reinforcement Learning vs. Other Learning Paradigms

- Supervised learning
  - Trains a model using a labeled dataset, which is a set of (input, label) pairs.
  - The model learns the relationship between input and its label (function approximation)
  - When an input is presented to the trained model, the model can predict its label
- Unsupervised learning
  - Trains a model using dataset without labels.
  - The trained model can organize data into clusters or find abnormal data
- Reinforcement learning
  - An agent interacts with the given environment by taking actions.
  - The agent learns to take good actions from feedback (reward) obtained from previous episodes.

# Fundamentals of Reinforcement Learning

---

## Markov Decision Process

# Markov Property and Markov Chain

- Markov Decision Process (MDP)
  - A mathematical framework for solving optimization problems
  - Used in RL to solve problems as well
- Markov property
  - "Future depends only on the present and not on the past."
  - a memoryless property
- Markov chain (Markov process)
  - Sequence of states that strictly obey the Markov property
  - A probabilistic model that solely depends on the current state to predict the next state (and not the previous states)
    - The future is conditionally independent of the past
  - e.g.) "today is cloudy, so it is likely to be rainy tomorrow."

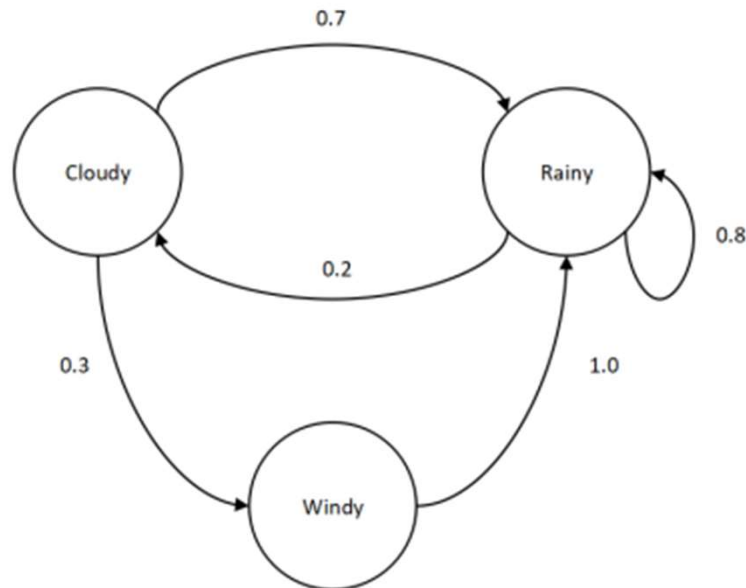
# Elements of Markov Chain

- States
  - A set of all possible states
  - e.g. [cloudy, windy, rainy]
- Transition probability
  - Probability of moving from one state to another

Current State	Next State	Transition Probability
Cloudy	Rainy	0.7
Cloudy	Windy	0.3
Rainy	Rainy	0.8
Rainy	Cloudy	0.2
Windy	Rainy	1.0

# Representation of Markov Chain

- State diagram of a Markov chain



- Transition matrix

	Cloudy	Rainy	Windy
Cloudy	0.0	0.7	0.3
Rainy	0.2	0.8	0.0
Windy	0.0	1.0	0.0

# Markov Reward Process (MRP)

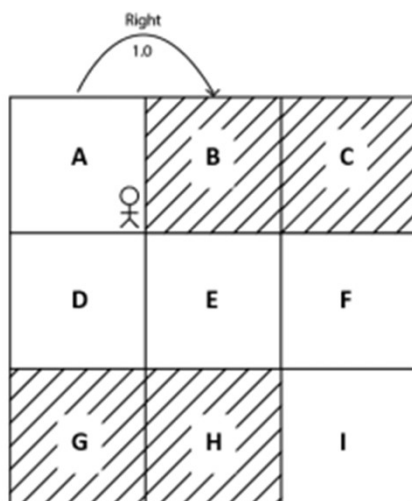
- Extension of Markov chain with the reward function
- Markov Chain = states + transition probability
- Markov Reward Process = states + transition probability + reward function
  - States:  $s$
  - Transition probability:  $P(s'|s)$
  - Reward function:  $R(s)$
- Reward function
  - A reward function defines reward obtained in each state
  - E.g.) reward in state cloudy, reward in state windy, reward in state rainy

# Markov Decision Process (MDP)

- Extension of Markov Reward Process
- Markov Chain = states + transition probability
- Markov Reward Process = states + transition probability + reward function
- Markov Decision Process = states + transition probability + reward function + actions
  - States:  $s$
  - Actions:  $a$
  - Transition probability:  $P(s'|s, a)$ 
    - Probability of moving to state  $s'$  when the agent takes action  $a$  in state  $s$ .
  - Reward function:  $R(s, a, s')$ 
    - Reward the agent obtains when moving from state  $s$  to  $s'$  while performing action  $a$ .
- Actions
  - A set of actions that our agent can perform in each state
  - An agent performs an action and moves from one state to another

# Formulating Grid World using MDP

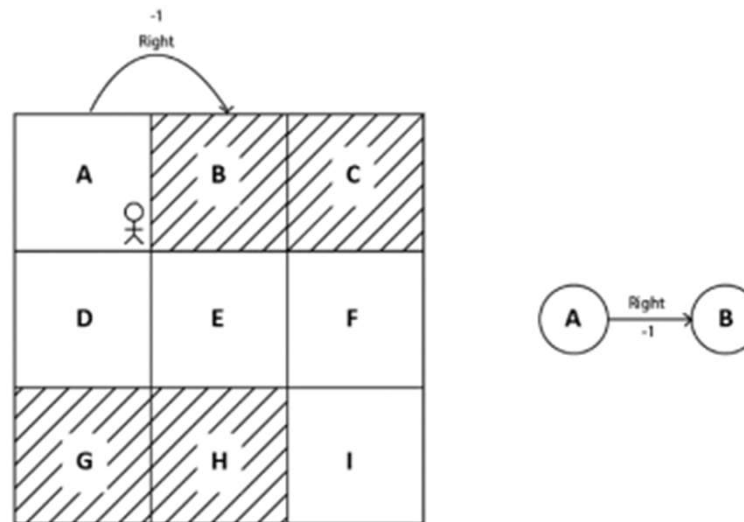
- States: the set of locations {A, B, ..., I}.
- Actions: the set of actions {down, right}.
- Transition Probability
  - E.g.)  $P(B|A, right)$  is the probability of the agent ends up in state B when taking action "right" in state A.
  - In our previous 3x3 Grid World example, the agent always moves to state B from state A when it moves "right".
    - $P(B|A, right) = 1.0$
    - In some environment, an action may probabilistically lead the agent to multiple states.





# Formulating Grid World using MDP

- Reward function
  - Reward for (current state, action, next state) triple
  - E.g.)  $R(A, right, B)$ : reward when the agent performs action "right" in state A, and moves to state B.
  - In the 3x3 Grid World, agent receives reward -1 when it moves to state B, because it is a shaded state.
    - $R(A, right, B) = -1.0$
    - Reward can be different when one of current state, action, next state is different.



# Fundamentals of Reinforcement Learning

---

## Concepts of RL

## Recap: Expectation (1)

- Suppose we have a random variable  $X$ .
- The random variable takes values based on a random experiment.
  - E.g.) rolling a dice, tossing a coin
- Suppose  $X$  is an outcome of rolling a fair dice.
- The possible outcomes are 1, 2, 3, 4, 5, and 6.
- Probability of occurrence for each value is  $1/6$ .

$X$	1	2	3	4	5	6
$P(x)$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$	$1/6$

- For a random variable  $X$ , its **expectation**  $E(X)$  is defined as:

$$E(X) = \sum_{i=1}^N x_i p(x_i)$$

- When rolling a dice, the expectation is 3.5.

## Recap: Expectation (2)

- We can define expectation of a function of a random variable X.
- Suppose we roll a dice, and we get a score of  $x^2$  when the outcome is  $x$ .
  - If the outcome of dice is 5, we get a score of 25.
  - What is the expected score we get, when we roll the dice?

X	1	2	3	4	5	6
f(x)	1	4	9	16	25	36
P(x)	1/6	1/6	1/6	1/6	1/6	1/6

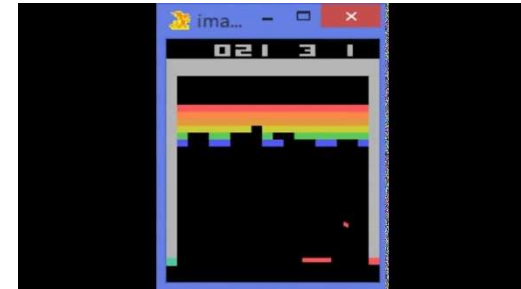
- The expectation of a function of a random variable is defined as:

$$\mathbb{E}_{x \sim p(x)}[f(X)] = \sum_{i=1}^N f(x_i)p(x_i)$$

- For the dice game, the expected score is 15.167.

# Action Space

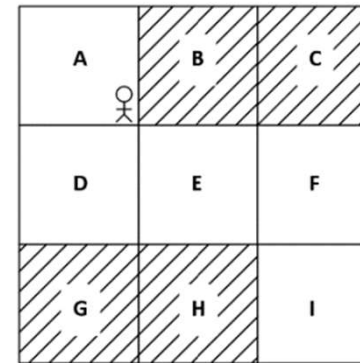
- The set of possible actions
  - E.g.) {right, down} in the Grid World.
  - E.g.) {right, left, stay, fire} in the Atari Breakout.
- Two types of **action spaces**
  - Discrete action space
  - Continuous action space
- Discrete action space
  - Action space consists of discrete actions, e.g.) up, down, left, right
- Continuous action space
  - Action space consists of continuous actions
  - E.g.) self-driving cars: speed of the car, number of degrees to rotate the wheel



# Policy

- A **policy** defines the agent's behavior in an environment
  - Action taken by the agent in each state

states	actions
A	right
B	right
C	down
F	down

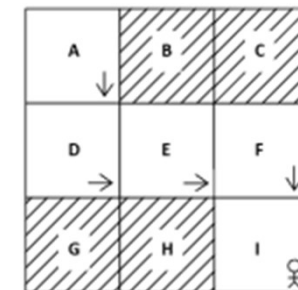


- When the agent first interacts with the environment, it uses a **random policy**.
  - The agent chooses a random action in every state.

- Optimal policy
  - The best policy in the environment
  - E.g.) The agent can reach the goal state without going through shaded states

Optimal Policy

State	Action
A	Down
D	Right
E	Right
F	Down



# Policy: Deterministic vs. Stochastic Policy

- **Deterministic Policy**

- The agent performs one particular action in a state.
- Denoted by  $\mu \rightarrow a_t = \mu(s_t)$
- If the agent moves down in state A, then:  $\mu(A) = \text{Down}$

- **Stochastic Policy**

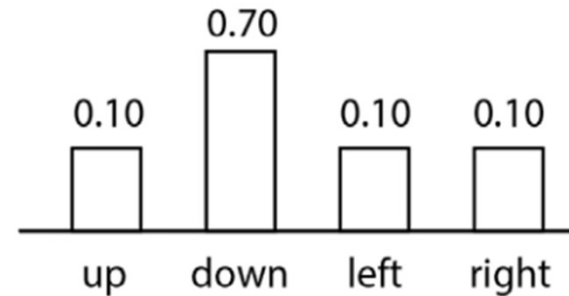
- In each state, the agent probabilistically selects one of the possible actions
- The agent may take different actions in a state in different episodes
- For example, in state A, the agent moves right 70% of the time, and moves down 30% of the time.
- The probability distribution over actions [right, down] in state A is [0.7, 0.3].
- The probability distribution is denoted as  $\pi$ .
- Expressions for stochastic policy

$$a_t \sim \pi(s_t) \quad \pi(a_t | s_t)$$

# Stochastic Policy: Categorical Policy vs. Gaussian Policy

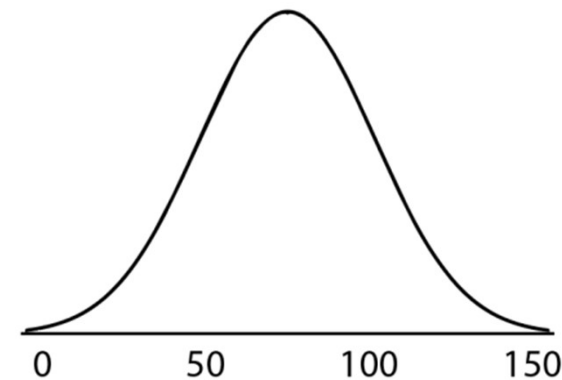
- **Categorical policy**

- action space is discrete
- E.g.) actions: up, down, left, right



- **Gaussian Policy**

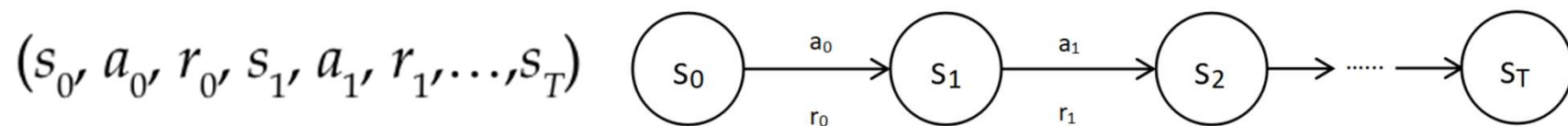
- action space is continuous
- E.g.) action: speed of a car, range of values: 0 to 150 kmph.
- Probability of selecting a speed





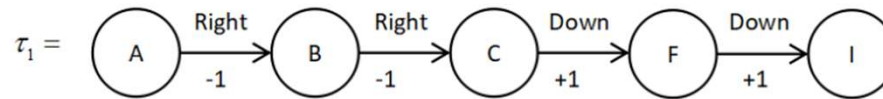
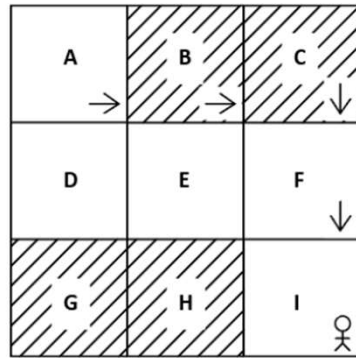
# Episode

- The agent interacts with the environment by performing some actions, starting from the initial state until it reaches the final (terminal) state.
- One iteration of this interaction is called an **episode**.
- E.g.) In a car racing video game, the player starts racing from the starting point and moves to the endpoint. It is considered an episode.
- An episode is often called a **trajectory**, and is denoted by  $\tau$ .
- In reinforcement learning, the agent goes through multiple episodes in order to find the optimal policy.
  - When playing a game, the user can learn good strategies to win the game by playing the game repeatedly.
- An episode can be represented by a sequence of (state, action, reward).

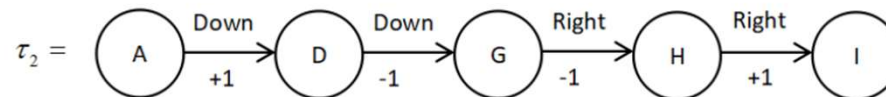
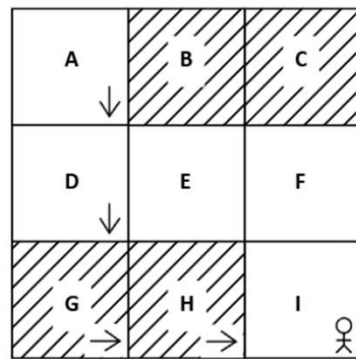


# Learning Optimal Policy from Episodes (1)

- Episode 1: with no experience, the agent uses a random policy.

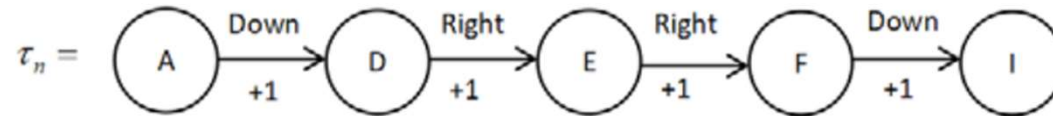
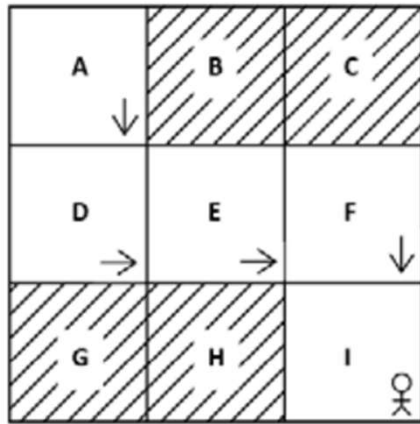


- Episode 2: since moving right in state A leads to a negative reward, the agent tries a different action this time and moves down.



## Learning Optimal Policy from Episodes (2)

- Episode n: over a series of episodes the agent learns the optimal policy.



# Episodic Task vs. Continuous Task

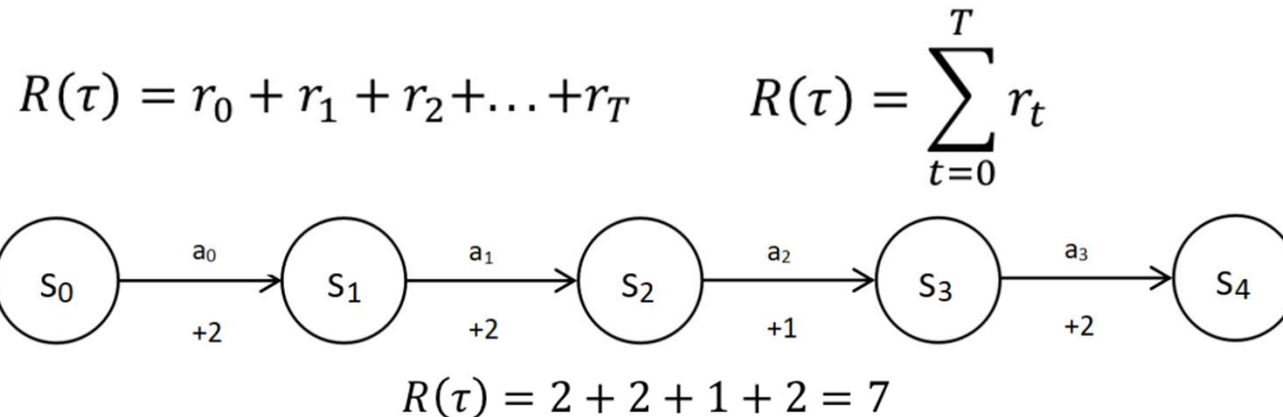
- Episodic Task
  - A task that has a terminal/final state.
  - Episodic tasks are tasks made up of episodes with terminal states.
  - E.g.) A car racing game where a car starts from the starting point and reaches the destination (final point).
- Continuous Task
  - Continuous tasks do not contain episodes, and they do not have any terminal state.
  - Continuous tasks are tasks that never end.
  - E.g.) A personal assistance robot

# Horizon

- Horizon is the time step until which the agent interacts with the environment.
- Finite horizon
  - The agent-environment interaction stops at a particular time step
  - In an episodic task, an agent interacts with the environment by starting from the initial state at time step  $t=0$  and reaches the final state at time step  $T$ .
- Infinite horizon
  - The agent-environment interaction never stops
  - In a continuous task, the agent-environment interaction does not terminate.

# Return

- A **return** can be defined as the sum of rewards obtained in an episode.
- Denoted by  $R$  or  $G$ .
- If the agent starts an episode from the initial state at time step 0 and reaches the final state at time step  $T$ ,



- If the task is a continuous task that does not end,

$$R(\tau) = r_0 + r_1 + r_2 + \dots + r_{\infty}$$

- The goal of agent is to learn an optimal policy that maximizes return.

# Discount Factor

- Let us consider a continuous task that does not end.
- The return of this task can be defined as:

$$R(\tau) = r_0 + r_1 + r_2 + \dots + r_\infty$$

- Suppose the rewards are all defined as nonnegative values.
- Then, the return sums to infinity regardless of what actions the agent takes.
- We cannot maximize return that goes to infinity.

- In order to deal with this problem, we introduce a discount factor  $\gamma$ .
- Discount factor is chosen from range  $[0, 1]$ .
- With the discount factor, the return is defined as:

$$R(\tau) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \dots + \gamma^n r_\infty \quad R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

- When  $\gamma < 1$ , the return is prevented from reaching infinity.

# Small Discount Factor vs. Large Discount Factor

- Small discount factor: discount factor close to 0.

- Suppose  $\gamma = 0.2$ , then return is:

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\ &= (0.2)^0 r_0 + (0.2)^1 r_1 + (0.2)^2 r_2 + \dots \\ &= (1)r_0 + (0.2)r_1 + (0.04)r_2 + \dots \end{aligned}$$

- More importance is given to the immediate reward.
  - When  $\gamma = 0$ , the return is just the immediate reward  $r_0$ . Agent does not learn.

- Large discount factor: discount factor close to 1.

- Suppose  $\gamma = 0.9$ , then return is:

$$\begin{aligned} R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\ &= (0.9)^0 r_0 + (0.9)^1 r_1 + (0.9)^2 r_2 + \dots \\ &= (1)r_0 + (0.9)r_1 + (0.81)r_2 + \dots \end{aligned}$$

- More importance is given to future rewards.
  - If  $\gamma = 1$ , the return becomes infinity in continuous tasks. Learning does not converge.

- The best discount factor depends on the task.



# Value Function

- The **value function**, or the state value function, denotes the **value of a state**.
- The value of a state is the return an agent would obtain starting from that state following policy  $\pi$ .

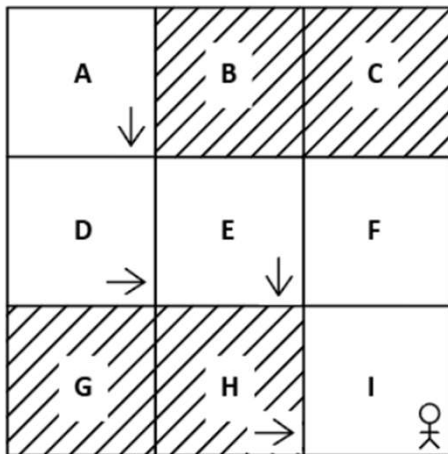
- Denoted as  $V(s)$

$$V^\pi(s) = [R(\tau) | s_0 = s]$$

- $s_0 = s$  means the starting state is  $s$ .

# Value Function: Example

- In the Grid World, suppose the agent's policy is shown in the figure below.
  - A: down, D: right, E: down, H: right
- Assume we set discount factor as 1.
- $V(A)$ : return of trajectory starting from state A  $\rightarrow 1 + 1 - 1 + 1 = 2$
- $V(D)$ : return of trajectory starting from state D  $\rightarrow 1 - 1 + 1 = 1$
- $V(E)$ : return of trajectory starting from state E  $\rightarrow -1 + 1 = 0$
- $V(H)$ : return of trajectory starting from F  $\rightarrow 1$ .
- $V(I)$ : terminal state  $\rightarrow 0$ .



# Value Function with Stochastic Policies

- When the policy is stochastic, actions in states are chosen probabilistically.
- Because of that, we use the **expected return**.

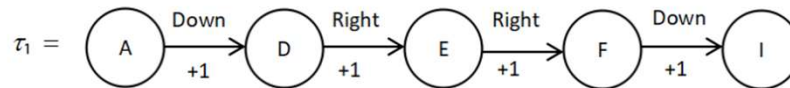
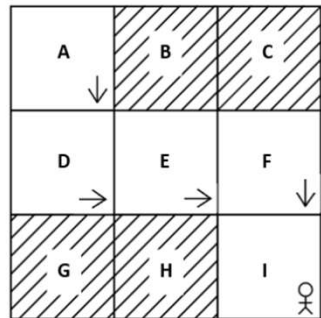
$$V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

# Value Function with Stochastic Policies: Example

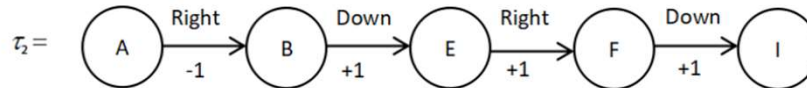
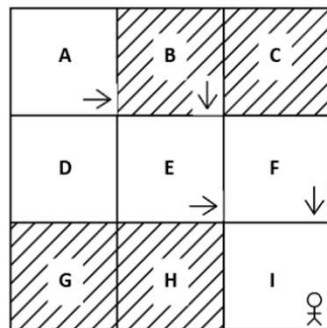
- Let us assume the actions in all states except state A are deterministic.
- In state A, the agent moves down 80% of the time, and moves right 20% of the time.
- What is the value of state A?

$$V^\pi(A) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = A] = \sum_i R(\tau_i) \pi(a_i | A)$$

$$\begin{aligned} &= R(\tau_1) \pi(\text{down} | A) + R(\tau_2) \pi(\text{right} | A) \\ &= 4(0.8) + 2(0.2) \\ &= 3.6 \end{aligned}$$



$$V(A) = R(\tau_1) = 1 + 1 + 1 + 1 = 4$$



$$V(A) = R(\tau_2) = -1 + 1 + 1 + 1 = 2$$

# Value Function and Policy

- The value function  $V(s)$  depends on the policy.
  - The value of a state varies based on the policy we choose, because the policy determines what actions are chosen in each state.
- The optimal value function  $V^*(s)$  yields the maximum value compared to all other functions
$$V^*(s) = \max_{\pi} V^{\pi}(s)$$
- The policy that maximizes value of all states is called the optimal policy  $\pi^*$ .
  - Proof of existence of the optimal policy in finite MDPs.
    - <https://towardsdatascience.com/why-does-the-optimal-policy-exist-29f30fd51f8c>
- Value table: a table that records values of all states
  - $V(s_1) > V(s_0)$  means that it is better to be in state  $s_1$  than  $s_0$ .
  - A state that has the maximum value is called the optimal state.

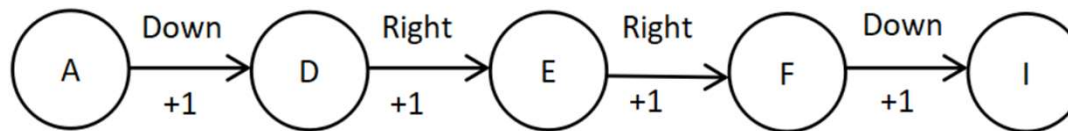
State	Value
$s_0$	7
$s_1$	11

# Q Function

- The value function defines value of a state.
- The Q function defines value of a state-action pair.
- The value of a state-action pair is the **return** the agent would obtain **starting from state  $s$**  and **performing action  $a$**  following **policy  $\pi$** .
- Denoted as  $Q(s, a)$

$$Q^\pi(s, a) = [R(\tau) | s_0 = s, a_0 = a]$$

- Suppose the agent follows the policy:



- $Q^\pi(A, down) = [R(\tau) | s_0 = A, a_0 = down] = 1 + 1 + 1 + 1 = 4$
- $Q^\pi(D, right) = [R(\tau) | s_0 = D, a_0 = right] = 1 + 1 + 1 = 3$
- We can compute Q value of all state-action pairs similarly.

# Q Function

- To address stochasticity, we use the expected return.

$$Q^{\pi}(s, a) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s, a_0 = a]$$

- Similar to the value function, the Q function depends on the policy.
- The optimal Q function: one that has the maximum Q value over other Q functions.

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$


- The optimal policy: the policy that gives the maximum Q value.
- Q table**: represents Q values of all possible state-action pairs

State	Action	Value
$s_0$	0	9
$s_0$	1	11
$s_1$	0	17
$s_1$	1	13

# Q Function and Optimal Policy

- The optimal policy can be extracted from the Q table.
  - For each state, select the action that yields the highest Q value.

Q Table					Optimal policy	
State	Action	Value				
$s_0$	0	9				
$s_0$	1	11				
$s_1$	0	17				
$s_1$	1	13				



State	Action
$s_0$	1
$s_1$	0

- Note: When we have the Q table, we can extract a policy based on the table.

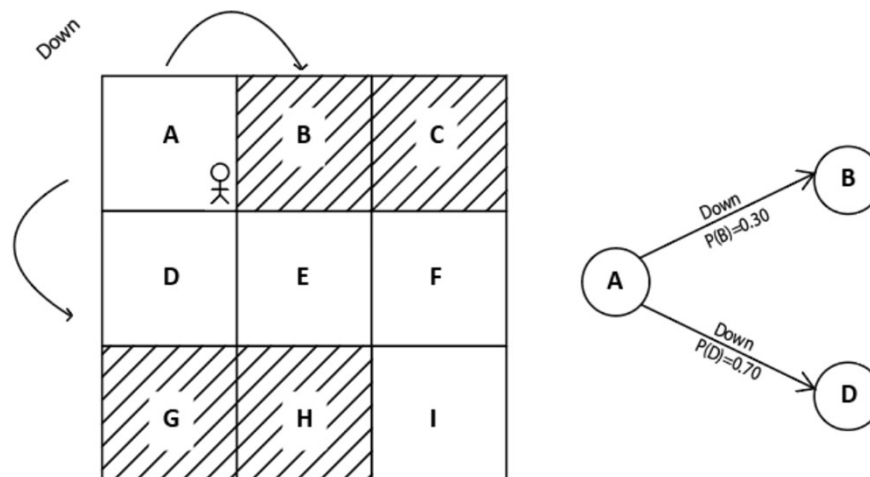


# Model-based Learning vs. Model-free Learning

- Here, the "model" means assumptions on the environment.
- Model-based learning
  - The environment dynamics is known.
    - Transition probability for a state-action pair.
    - Reward function is known.
  - For example, in a game of Go, the rules of the game is known.
    - The next state is determined based on the action (putting a stone on the board)
    - The rules on winning the game is also known.
- Model-free learning
  - The agent does not know the environment dynamics.

# Different Types of Environments

- The environment is the world of the agent.
  - The agent lives/stays within the environment.
- Deterministic vs. Stochastic environments
  - Deterministic environment: when an agent performs action  $a$  in state  $s$ , its next state is always the same.
  - Stochastic environment: when an agent performs action  $a$  in state  $s$ , its next state is determined probabilistically.
    - When the agent moves "down" in state A, the agent successfully arrive at state D 70% of the time, but it fails and ends up in state B 30% of the time.



# Different Types of Environments

- Discrete vs. Continuous environments
  - Discrete environment: action space is discrete. E.g.) [up, down, left, right]
  - Continuous environment: action space is continuous. E.g.) car's speed, rotating the wheel
- Episodic vs. Non-episodic environments
  - Episodic environment: the agent's current action does not affect future actions
    - e.g.) asking a question to a bot
  - Non-episodic environment: the agent's current action affects future actions
    - Also called sequential environment
    - e.g.) chessboard
- Single vs. Multi-agent environments
  - Single-agent environment: The environment consists of a single agent.
    - e.g.) a game that is played by a single player
  - Multi-agent environment: Multiple agents are in the environment
    - e.g.) playing a soccer match

# Different Types of Environments

- Fully Observable vs. Partially Observable environments
  - Fully observable environment: driving a car on the road
  - Partially observable environment: playing card games such as Poker or Blackjack
- Static vs. Dynamic environments
  - Static environment: the environment does not change while the agent is sensing the environment
    - e.g.) cleaning a room using a robot cleaner
  - Dynamic environment: the environment changes when the agent is sensing the environment
    - e.g.) playing a soccer match

# Applications of Reinforcement Learning

- Manufacturing
  - Intelligent robots are trained using RL to place objects in the right position.
- Dynamic pricing
  - Based on demand and supply, an RL agent learns to determine price of products with the goal of maximizing revenue.
- Inventory management
  - RL is used for tasks such as supply chain management, demand forecasting, and handling warehouse operations (distributing products in warehouses.)
- Recommendation system
  - The RL agent learns the user's preference and recommends books, songs, or movies.
- Neural architecture search
  - RL is used to find the best neural architecture for a given task with the goal of maximizing accuracy
- Natural language processing
  - RL is used in NLP tasks such as abstractive text summarization and chatbots.
- Finance
  - RL is used in financial portfolio management, which is the process of constant redistribution of a fund into different financial products

## End of Chapter

- Do you remember these keywords?
- agent
- environment
- state
- action
- reward
- action space
- policy
- episode
- episodic and continuous task
- horizon
- return
- discount factor
- value function
- Q function
- Model-based and model-free learning
- deterministic and stochastic environment

# Homework: Install Python Environment + Gym Toolkit

- The Gym toolkit is a software developed and maintained by OpenAI.
  - <https://gym.openai.com/>
  - The toolkit provides simulated environments for training RL agents.
- Until next class, prepare a python environment and install the Gym toolkit on your computer.
- IDE: Visual Studio Code
  - <https://code.visualstudio.com/>
  - Install Python extension
- Virtual Environment Manager: Anaconda
  - <https://www.anaconda.com/products/individual-d>

# Homework: Install Python Environment + Gym Toolkit

- Goal: you can successfully import the gym library in your python program.
- Try running the following code and see if you get a printout of the Frozen Lake environment.

```
[1] ▶ M4  
  
import gym  
env = gym.make("FrozenLake-v0")  
state = env.reset()  
env.render()  
  
SFFF  
FHFH  
FFFF  
HFFG
```



End of Class

---

Questions?

Email: [jso1@sogang.ac.kr](mailto:jso1@sogang.ac.kr)