대학교육개발센터협의회 하계워크숍

AI IN TEACHING & LEARNING

AI 시대의 교수법

슬라이드 생성부터 수업자료 구성까지

전민석 · CTL, DGIST

PART 1 · 교수자

Claude Code로
강의 슬라이드 자동 생성

- 구조화된 프롬프트 하나로 강의 자료 초안
- Claude Code × Beamer LaTeX

PART 2 · 학습자

VS Code + Claude Code로
수업자료 학습

- 강의 슬라이드로부터 나만의 AI 튜터 생성하기
- 개념 설명 · 자가 진단

PART 1

01

Claude Code로 강의 슬라이드 만들기

강의 자료의 초안을 몇 분 만에 자동으로 생성하기

기본 세팅, 단 4단계

1. **Claude Code 설치**
   npm 한 줄이면 끝
2. **강의 폴더에서 실행**
   폴더 생성 후 `claude` 실행
3. **구조화된 프롬프트 입력**
   Context · Role · Command · Format
4. **slide.tex → PDF 컴파일**
   pdflatex 또는 Overleaf

Terminal

```
# 1. Claude Code 설치
npm install -g @anthropic-ai/claude-code

# 2. 강의 폴더 생성 후 claude 실행
mkdir Course && cd Course
claude

# 3. 구조화된 프롬프트 입력
Context: I need lecture notes & slides for {Course}.
Role: You are an instructor authoring materials
      using Beamer LaTeX.
Command: 1) roadmap.md  2) topics/*.md
         3) slides/Topic1/ Beamer (.tex)
Format: roadmap.md · topics/topicN.md
        · slides/Topic1/slide.tex

# 4. LaTeX → PDF 컴파일
pdflatex slides/Topic1/slide.tex
```

프롬프트는 네 가지로 구조화

Context
어떤 과목의, 누구를 위한 자료인지 — 배경과 수준을 알려줍니다.

Role
"Beamer LaTeX로 자료를 만드는 교수자" 처럼 역할을 부여합니다.

Command
로드맵 → 토픽별 노트 → 슬라이드 순서로, 할 일을 단계로 지시합니다.

Format
결과물의 파일 구조(roadmap.md · topics/ · slides/)를 명시합니다.

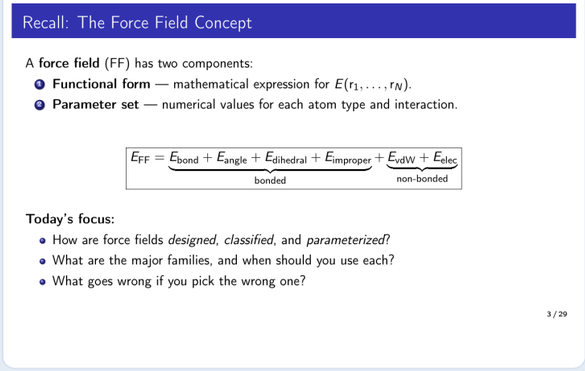
실제 적용 사례

경영전문대학원

확률론 부트캠프

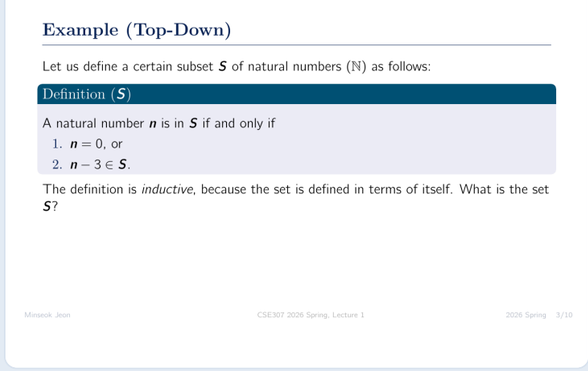
화학물리학과

계산화학



전기전자컴퓨터공학과

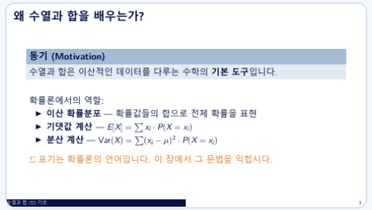
프로그래밍 언어

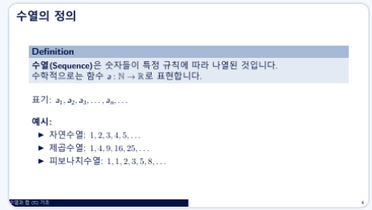


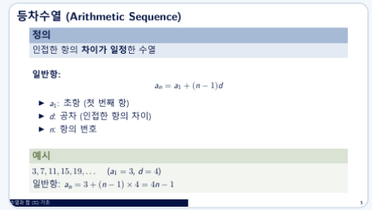
실제 생성물

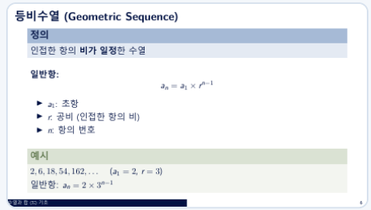

1. 수열과 합(Σ) 기초
Sequences and Summation — 확률론 부트캠프
DGIST Graduate School of Technology & Innovation Management
Probability with Mathematical Foundations

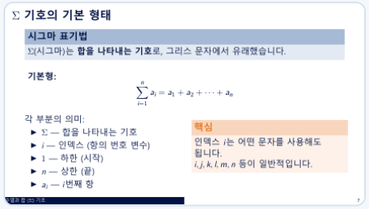

Outline
Introduction
수열 (Sequence)
합의 표기: Σ
합의 성질 (연산 규칙)
이중합 (Double Σ)
연습 문제
확률론과의 연결

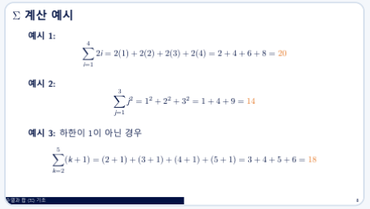

왜 수열과 합을 배우는가?
동기 (Motivation)
수열과 합은 이산적인 데이터를 다루는 수학의 기본 도구입니다.
확률론에서의 역할:
이산 확률분포 — 확률값들의 합으로 전체 확률을 표현
기댓값 계산 — E[X] = Σ x_i · P(X = x_i)
분산 계산 — Var(X) = Σ(x_i − μ)² · P(X = x_i)
Σ 표기는 확률론의 언어입니다. 이 장에서 그 문법을 익힙시다.


수열의 정의
Definition
수열(Sequence)은 수자들이 특정 규칙에 따라 나열된 것입니다.
수학적으로는 함수 a : ℕ → ℝ로 표현합니다.
표기: a₁, a₂, a₃, . . . , aₙ, . . .
예시:
자연수열: 1, 2, 3, 4, 5, . . .
제곱수열: 1, 4, 9, 16, 25, . . .
피보나치수열: 1, 1, 2, 3, 5, 8, . . .


등차수열 (Arithmetic Sequence)
정의
인접한 항의 차이가 일정한 수열
일반항:
aₙ = a₁ + (n − 1)d
a₁: 초항 (첫 번째 항)
d: 공차 (인접한 항의 차이)
n: 항의 번호
예시
3, 7, 11, 15, 19, . . . (a₁ = 3, d = 4)
일반항: aₙ = 3 + (n − 1) × 4 = 4n − 1

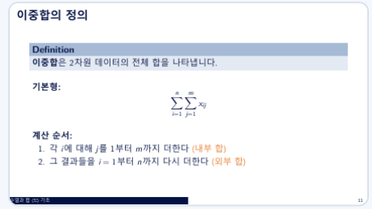

등비수열 (Geometric Sequence)
정의
인접한 항의 비가 일정한 수열
일반항:
aₙ = a₁ × r^(n−1)
a₁: 초항
r: 공비 (인접한 항의 비)
n: 항의 번호
예시
2, 6, 18, 54, 162, . . . (a₁ = 2, r = 3)
일반항: aₙ = 2 × 3^(n−1)

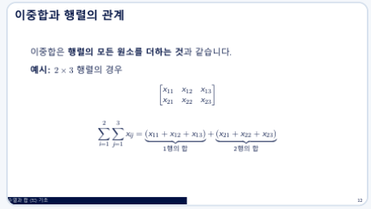

Σ 기호의 기본 형태
시그마 표기법
Σ(시그마)는 합을 나타내는 기호로, 그리스 문자에서 유래했습니다.
기본형:
Σ_{i=1}^{n} a_i = a₁ + a₂ + ⋯ + aₙ
각 부분의 의미:
Σ — 합을 나타내는 기호
i — 인덱스 (항의 번호 변수)
1 — 하한 (시작)
n — 상한 (끝)
a_i — i번째 항
핵심
인덱스 i는 어떤 문자를 사용해도 됩니다.
i, j, k, l, m, n 등이 일반적입니다.

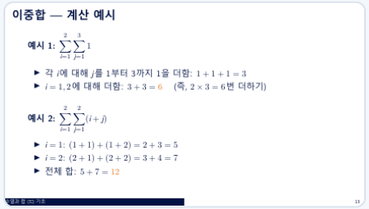

Σ 계산 예시
예시 1:
Σ_{i=1}^{4} 2i = 2(1) + 2(2) + 2(3) + 2(4) = 2 + 4 + 6 + 8 = 20
예시 2:
Σ_{j=1}^{3} j² = 1² + 2² + 3² = 1 + 4 + 9 = 14
예시 3: 하한이 1이 아닌 경우
Σ_{k=2}^{5} (k + 1) = (2 + 1) + (3 + 1) + (4 + 1) + (5 + 1) = 3 + 4 + 5 + 6 = 18


선형성 (Linearity)
상수배의 법칙
Σ_{i=1}^{n} c · a_i = c · Σ_{i=1}^{n} a_i
각 항에 상수 c를 곱한 것의 합 = 합 전체에 c를 곱한 것
덧셈의 법칙
Σ_{i=1}^{n} (a_i + b_i) = Σ_{i=1}^{n} a_i + Σ_{i=1}^{n} b_i
두 수열의 항을 더한 것의 합 = 각 수열의 합을 더한 것

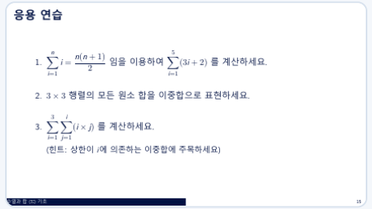

선형성 — 계산 예시
상수배의 법칙:
Σ_{i=1}^{3} 2i = 2 · Σ_{i=1}^{3} i = 2 · (1 + 2 + 3) = 2 · 6 = 12
덧셈의 법칙:
Σ_{i=1}^{2} (i + i²) = Σ_{i=1}^{2} i + Σ_{i=1}^{2} i² = (1 + 2) + (1 + 4) = 3 + 5 = 8
복합 활용:
Σ_{i=1}^{4} (3i − 2) = 3 · Σ_{i=1}^{4} i − Σ_{i=1}^{4} 2 = 3 · 10 − 4 · 2 = 30 − 8 = 22

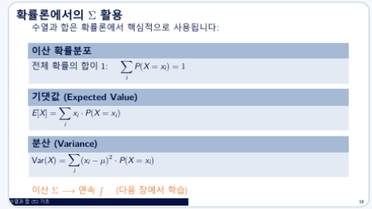

이중합의 정의
Definition
이중합은 2차원 데이터의 전체 합을 나타냅니다.
기본형:
Σ_{i=1}^{n} Σ_{j=1}^{m} x_ij
계산 순서:
1. 각 i에 대해 j를 1부터 m까지 더한다 (내부 합)
2. 그 결과들을 i = 1부터 n까지 다시 더한다 (외부 합)


이중합과 행렬의 관계
이중합은 행렬의 모든 원소를 더하는 것과 같습니다.
예시: 2 × 3 행렬의 경우
[x11 x12 x13; x21 x22 x23]
Σ_{i=1}^{2} Σ_{j=1}^{3} x_ij = (x11 + x12 + x13) + (x21 + x22 + x23)
1행의 합
2행의 합


이중합 — 계산 예시
예시 1: Σ_{i=1}^{2} Σ_{j=1}^{3} 1
각 i에 대해 j를 1부터 3까지 1을 더함: 1 + 1 + 1 = 3
i = 1, 2에 대해 더함: 3 + 3 = 6 (즉, 2 × 3 = 6번 더하기)
예시 2: Σ_{i=1}^{2} Σ_{j=1}^{2} (i + j)
i = 1: (1 + 1) + (1 + 2) = 2 + 3 = 5
i = 2: (2 + 1) + (2 + 2) = 3 + 4 = 7
전체 합: 5 + 7 = 12


기본 연습
다음을 계산하세요:
1. Σ_{i=1}^{4} 2i = ?
2. Σ_{i=1}^{2} Σ_{j=1}^{3} 1 = ? (총 몇 번 더하는가?)
3. 등차수열 5, 8, 11, 14, . . . 의 10번째 항은?
4. 등비수열 3, 6, 12, 24, . . . 의 6번째 항은?


응용 연습
1. Σ_{i=1}^{n} i = n(n+1)/2 임을 이용하여 Σ_{i=1}^{5} (3i + 2) 를 계산하세요.
2. 3 × 3 행렬의 모든 원소 합을 이중합으로 표현하세요.
3. Σ_{i=1}^{3} Σ_{j=1}^{i} (i × j) 를 계산하세요.
(힌트: 상한이 j에 의존하는 이중합에 주목하세요)


확률론에서의 Σ 활용
수열과 합은 확률론에서 핵심적으로 사용됩니다:
이산 확률분포
전체 확률의 합이 1: Σ_i P(X = x_i) = 1
기댓값 (Expected Value)
E[X] = Σ_i x_i · P(X = x_i)
분산 (Variance)
Var(X) = Σ_i (x_i − μ)² · P(X = x_i)
이산 Σ ⟶ 연속 ∫ (다음 장에서 학습)


Key Takeaways
수열은 규칙에 따라 나열된 수의 열 (등차, 등비)
Σ는 합을 간결하게 표현하는 기호
합의 선형성으로 복잡한 계산을 단순화
이중합은 2차원 데이터의 전체 합을 표현
확률론의 기댓값, 분산 등에서 Σ가 핵심

Thank you
Questions?

실제 생성물

Overview of Force Fields for Classical Simulations

Computational Chemistry for Undergraduates

Instructor

March 15, 2026

Outline

Recall: The Force Field Concept

A **force field** (FF) has two components:

1. **Functional form** — mathematical expression for $E(r_1, \ldots, r_N)$.
2. **Parameter set** — numerical values for each atom type and interaction.

$$E_{\text{FF}} = \underbrace{E_{\text{bond}} + E_{\text{angle}} + E_{\text{dihedral}} + E_{\text{improper}}}_{\text{bonded}} + \underbrace{E_{\text{vdW}} + E_{\text{elec}}}_{\text{non-bonded}}$$

Today's focus:

- How are force fields *designed*, *classified*, and *parameterized*?
- What are the major families, and when should you use each?
- What goes wrong if you pick the wrong one?

Atom Types: The Building Blocks

Force fields do *not* treat each atom individually. Instead, atoms with similar chemical environments share an **atom type**.

Atom Type	Description	FF Example
CT	sp^3 carbon	AMBER
CA	Aromatic carbon	AMBER
C	Carbonyl carbon (sp^2)	AMBER
N	Amide nitrogen	AMBER
OH	Hydroxyl oxygen	AMBER
HC	H bonded to carbon	AMBER
HO	H bonded to oxygen	AMBER

Key idea — transferability: Parameters for a C–C single bond should work in ethane, butane, cyclohexane, *and* a protein backbone.

Class I, II, and III Force Fields

Class	Functional Form	Examples
Class I	Harmonic bonds/angles, Fourier torsions, fixed-charge Coulomb + LJ	AMBER, CHARMM, OPLS-AA, GROMOS
Class II	Add cross-coupling terms (stretch-bend, bend-torsion, etc.); higher-order bonded terms	CFF, COMPASS, MMFF94, PCFF
Class III	Polarizable electrostatics (induced dipoles, Drude oscillators, fluctuating charges)	AMOEBA, Drude (CHARMM), FQ

Trade-offs:

- Class I: fastest, most transferable, most validated.
- Class II: better vibrational frequencies and conformational energies.
- Class III: captures polarization, important for ions, interfaces.

All-Atom vs. United-Atom vs. Coarse-Grained

Resolution	H atoms	Speed	Example
All-atom (AA)	Explicit	$1\times$	AMBER ff19SB
United-atom (UA)	Merged into heavy atoms	~ 3–$5\times$	GROMOS, TraPPE
Coarse-grained (CG)	~ 4 atoms $\rightarrow$ 1 bead	~ 100–$1000\times$	Martini 3

- **UA**: Non-polar H atoms (e.g., $-CH_2-$, $-CH_3$) are absorbed into the parent carbon. Reduces atom count by $\sim 50\%$.
- **CG**: Sacrifices atomistic detail for access to μs–ms timescales (membranes, polymers, self-assembly).
- **Multiscale backmapping**: CG $\rightarrow$ AA to recover detail.

Bond Stretching

Harmonic (Class I — most common):

$$E_{\text{bond}} = \sum_{\text{bonds}} \frac{1}{2} k_b (r - r_0)^2$$

Morse (better for large deformations):

$$E_{\text{Morse}} = D_e \left[1 - e^{-\alpha(r - r_0)}\right]^2, \quad \alpha = \sqrt{k_b / 2D_e}$$

Bond	r_0 (Å)	k_b (kcal/mol/Å^2)	Period (fs)
C–C	1.53	~300	~20
C=C	1.34	~500	~16
C–H	1.09	~340	~10
O–H	0.96	~500	~9

O–H / C–H vibrations are fastest $\Rightarrow$ they limit the time step!
Constrain with SHAKE/LINCS $\Rightarrow$ use $\Delta t = 2$ fs.

Angle Bending and Torsions

Angle bending (harmonic):

$$E_{\text{angle}} = \sum_{\text{angles}} \frac{1}{2} k_\theta (\theta - \theta_0)^2$$

Proper torsion (Fourier series):

$$E_{\text{dihedral}} = \sum_{\text{dihedrals}} \sum_{n} \frac{V_n}{2} \left[1 + \cos(n\phi - \gamma_n)\right]$$

- $n = 1, 2, 3$ terms capture rotational barriers.
- **Critical for conformational preferences:** gauche vs. anti, cis vs. trans, ϕ/ψ Ramachandran angles.

Improper torsion (harmonic out-of-plane):

$$E_{\text{improper}} = \frac{1}{2} k_\omega (\omega - \omega_0)^2$$

Enforces planarity at sp^2 centers (aromatic rings, amide bonds).

Cross Terms (Class II)

Class II force fields add coupling terms:

- **Stretch–bend**: Bond length affects optimal angle (and vice versa).

$$E_{\text{sb}} = k_{r\theta} (r - r_0)(\theta - \theta_0)$$

- **Stretch–torsion**: Bond length affects torsional barrier.
- **Bend–bend**: Adjacent angles are coupled.

Benefit: Better vibrational frequencies, more accurate geometries.
Cost: More parameters $\Rightarrow$ harder to parameterize and transfer.

Examples: CFF, COMPASS (widely used in materials science), MMFF94 (drug design).

Van der Waals: Lennard-Jones and Alternatives

Lennard-Jones 12–6 (standard):

$$E_{\text{LJ}}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Buckingham (exp-6) — more physical repulsion:

$$E_{\text{Buck}}(r) = A\, e^{-Br} - \frac{C}{r^6}$$

Mie potential (generalized LJ):

$$E_{\text{Mie}}(r) = C_\varepsilon \left[\left(\frac{\sigma}{r}\right)^{n} - \left(\frac{\sigma}{r}\right)^{m}\right], \quad n > m$$

- SAFT-γ Mie force field uses variable n, m for chemical engineering applications.
- **Combining rules** (Lorentz–Berthelot, geometric, Waldman–Hagler) determine mixed parameters σ_{ij}, ε_{ij}.

Electrostatics: Charges and Beyond

Fixed point charges (Class I):

$$E_{\text{elec}} = \sum_{i<j} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$

Charge assignment methods:

Method	Used in	How
RESP	AMBER	Fit to QM electrostatic potential with restraints
AM1-BCC	GAFF	Semi-empirical + bond charge correction
CGenFF	CHARMM	Analogy + QM fitting via ParamChem
CM5	Various	Hirshfeld charges + empirical mapping

Beyond fixed charges (Class III):

- **Induced dipoles**: AMOEBA — each atom has a polarizability α_i.
- **Drude oscillators**: Auxiliary charged particle on a spring.

1–4 Interactions and Exclusions

Atoms connected by 1, 2, or 3 bonds: **exclude** non-bonded interactions (already handled by bonded terms).

Atoms separated by exactly **3 bonds (1–4 pairs)**: special treatment to avoid double-counting.

Force Field	LJ 1–4 scale	Elec 1–4 scale
AMBER	1/2	1/1.2 (≈ 0.833)
CHARMM	Special 1–4 LJ params	1.0 (full)
OPLS-AA	1/2	1/2
GROMOS	1.0 (full, but excluded pairs differ)	1.0

Warning

Mixing force fields with different 1–4 rules produces **incorrect energies**. Never mix AMBER charges with OPLS LJ parameters!

Long-Range Electrostatics: Ewald and PME

Coulomb interactions decay as $1/r$ — **too slowly** to truncate.

Simple cutoff: produces severe artifacts (pressure jumps, incorrect solvation free energies, broken energy conservation).

Ewald summation: Split Coulomb sum into:

$$E_{\text{elec}} = \underbrace{E_{\text{short-range}}}_{\text{real space (erfc)}} + \underbrace{E_{\text{long-range}}}_{\text{reciprocal space (FFT)}} + E_{\text{self}}$$

Particle Mesh Ewald (PME): Uses FFT for the reciprocal-space sum.

- Cost: $O(N \log N)$ instead of $O(N^2)$.
- **Standard in all modern MD codes** (GROMACS, AMBER, NAMD, OpenMM).
- Parameters to set: real-space cutoff (~ 10 Å), grid spacing (~ 1 Å), interpolation order (usually 4).

AMBER Force Fields

Assisted Model Building with Energy Refinement

Version	Domain
ff14SB / ff19SB	Proteins
OL3 / OL15	RNA / DNA
Lipid17	Lipid bilayers
GLYCAM06	Carbohydrates
GAFF / GAFF2	General small molecules

Key features:

- RESP charges from HF/6-31G* electrostatic potential.
- Lorentz–Berthelot combining rules.
- 1–4 scaling: LJ = 1/2, elec = 1/1.2.
- Compatible water models: TIP3P, OPC, SPC/E.
- Runs in: AMBER, GROMACS, OpenMM, NAMD.

CHARMM Force Fields

Chemistry at HARvard Macromolecular Mechanics

Version	Domain
CHARMM36m	Proteins (with CMAP correction)
CHARMM36	Lipids, nucleic acids, carbohydrates
CGenFF	General small molecules (ParamChem)
CHARMM-Drude	Polarizable version

Key features:

- **CMAP**: 2D dihedral correction map for backbone ϕ/ψ — significantly improves protein secondary structure.
- Special 1–4 LJ parameters (no simple scaling factor).
- Charges fitted to QM interaction energies with water.
- Compatible water: TIP3P (modified), SPC/E.

OPLS Force Fields

Optimized Potentials for Liquid Simulations

Version	Notes
OPLS-AA	All-atom; parameterized against liquid-state properties
OPLS3e	Extended coverage of drug-like molecules (Schrödinger)
OPLS4	Latest generation; includes many heterocycles

Key features:

- Designed to reproduce **liquid densities, heats of vaporization, and solvation free energies**.
- 1–4 scaling: both LJ and elec = 1/2.
- Geometric combining rules.
- Particularly strong for small organic molecules and drug design.
- Compatible water: TIP3P, TIP4P.

GROMOS Force Fields

GROningen MOlecular Simulation

Version	Notes
54A7 / 54A8	Standard biomolecular FFs
2016H66	Latest; improved lipid and carbohydrate parameters
ATB	Automated Topology Builder for small molecules

Key features:

- **United-atom** option: non-polar H merged into parent C.
- ~ 3–$5\times$ faster than all-atom FFs.
- Parameterized against condensed-phase thermodynamic data.
- SPC water model (standard).
- Strong in European computational chemistry community.

Comparison at a Glance

Feature	AMBER	CHARMM	OPLS	GROMOS
Class	I	I (+CMAP)	I	I
Resolution	AA	AA	AA	UA / AA
Charges	RESP	QM + water fit	CM1A/CM5	Empirical
LJ combining rule	L–B	L–B (special 1–4)	Geometric	Geometric
Elec 1–4 scale	0.833	1.0	0.5	1.0
Backbone correction	—	CMAP	—	—
Water model	TIP3P, OPC	mTIP3P	TIP4P	SPC

Critical Rule

Never mix parameters from different force field families! Charges, LJ parameters, 1–4 rules, and combining rules are *co-optimized* — mixing them breaks the balance.

실제 생성물

Lecture 4 — Recursive and Higher-Order Programming

CSE307: Programming Languages

Minseok Jeon

2026 Spring

Why Recursive and Higher-Order Programming?

Recursion and higher-order functions are essential in functional programming:

- Recursion is used instead of loops and provides a powerful problem-solving method.
- Higher-order functions provide a powerful means for abstractions (i.e. the capability of combining simple ideas to form more complex ideas).

The Power of Recursive Thinking

Quiz) Describe an algorithm to draw the following pattern:

The Power of Recursive Thinking

Quiz) Describe an algorithm to draw the following pattern:

```
function sierpinski(x, y, size, depth):
    if depth == 0:
        draw_triangle(x, y, size)
    else:
        half = size / 2
        sierpinski(x, y, half, depth - 1)
        sierpinski(x + half, y, half, depth - 1)
        sierpinski(x + half/2, y + half*sqrt(3)/2, half, depth - 1)
```

Recursive Problem-Solving Strategy

- If the problem is sufficiently small, directly solve the problem.
- Otherwise,
  1. Decompose the problem to smaller ones with the same structure as original.
  2. Solve each of those smaller problems.
  3. Combine the results to get the overall solution.

Example: list length

- If the list is empty, the length is 0.
- Otherwise,
  1. The list can be split into its head and tail.
  2. Compute the length of the tail.
  3. The overall solution is the length of the tail plus one.

Example: list length

- If the list is empty, the length is 0.
- Otherwise,
  1. The list can be split into its head and tail.
  2. Compute the length of the tail.
  3. The overall solution is the length of the tail plus one.

```
# length [];;
- : int = 0
# length [1;2;3];;
- : int = 3

let rec length l =
  match l with
  | [] -> 0
  | hd::tl -> 1 + length tl
```

Exercise 1: append

Write a function that appends two lists:

```
# append [1; 2; 3] [4; 5; 6; 7];;
- : int list = [1; 2; 3; 4; 5; 6; 7]
# append [2; 4; 6] [8; 10];;
- : int list = [2; 4; 6; 8; 10]

let rec append l1 l2 =
```

Exercise 2: reverse

Write a function that reverses a given list:

```
val reverse : 'a list -> 'a list = <fun>
# reverse [1; 2; 3];;
- : int list = [3; 2; 1]
# reverse ["C"; "Java"; "OCaml"];;
- : string list = ["OCaml"; "Java"; "C"]

let rec reverse l =
```

Exercise 3: nth-element

Write a function that computes *n*th element of a list:

```
# nth [1;2;3] 0;;
- : int = 1
# nth [1;2;3] 1;;
- : int = 2
# nth [1;2;3] 2;;
- : int = 3
# nth [1;2;3] 3;;
Exception: Failure "list is too short".

let rec nth l n =
  match l with
  | [] -> raise (Failure "list is too short")
  | hd::tl -> (* ... *)
```

Exercise 4: remove-first

Write a function that removes the first occurrence of an element from a list:

```
# remove_first 2 [1; 2; 3];;
- : int list = [1; 3]
# remove_first 2 [1; 2; 3; 2];;
- : int list = [1; 3; 2]
# remove_first 4 [1;2;3];;
- : int list = [1; 2; 3]
# remove_first [1; 2] [[1; 2; 3]; [1; 2]; [2; 3]];;
- : int list list = [[1; 2; 3]; [2; 3]]

let rec remove_first a l =
  match l with
  | [] -> []
  | hd::tl -> (* ... *)
```

Exercise 5: insert

Write a function that inserts an element to a sorted list:

```
# insert 2 [1;3];;
- : int list = [1; 2; 3]
# insert 1 [2;3];;
- : int list = [1; 2; 3]
# insert 3 [1;2];;
- : int list = [1; 2; 3]
# insert 4 [];;
- : int list = [4]

let rec insert a l =
  match l with
  | [] -> [a]
  | hd::tl -> (* ... *)
```

Exercise 6: insertion sort

Write a function that performs insertion sort:

```
let rec sort l =
  match l with
  | [] -> []
  | hd::tl -> insert hd (sort tl)
```

cf) Compare with "C-style" non-recursive version:

```
for (c = 1 ; c <= n - 1; c++) {
 d = c;
 while ( d > 0 && array[d] < array[d-1]) {
   t          = array[d];
   array[d]   = array[d-1];
   array[d-1] = t;
   d--;
 }
}
```

cf) Imperative vs. Functional Programming

- Imperative programming focuses on describing **how** to accomplish the given task:

```
int factorial (int n) {
  int i; int r = 1;
  for (i = 0; i < n; i++)
    r = r * i;
  return r;
}
```

  Imperative languages encourage to use statements and loops.

- Functional programming focuses on describing **what** the program must accomplish:

```
let rec factorial n =
  if n = 0 then 1 else n * factorial (n-1)
```

  Functional languages encourage to use expressions and recursion.

Is Recursion Expensive?

- In C and Java, we are encouraged to avoid recursion because function calls consume additional memory.

```
void f() { f(); }          /* stack overflow */
```

- This is not true in functional languages. The same program in ML iterates forever:

```
let rec f () = f ()
```

Tail-Recursive Functions

More precisely, *tail-recursive functions* are not expensive in ML. A recursive call is a tail call if there is nothing to do after the function returns.

-

```
let rec last l =
  match l with
  | [a] -> a
  | _::tl -> last tl
```

-

```
let rec factorial a =
    if a = 1 then 1
    else a * factorial (a - 1)
```

Languages like ML, Scheme, Scala, and Haskell do *tail-call optimization*, so that tail-recursive calls do not consume additional amount of memory.

cf) Transforming to Tail-Recursive Functions

Non-tail-recursive factorial:

```
let rec factorial a =
  if a = 1 then 1
  else a * factorial (a - 1)
```

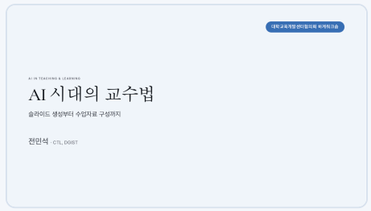
Tail-recursive version:

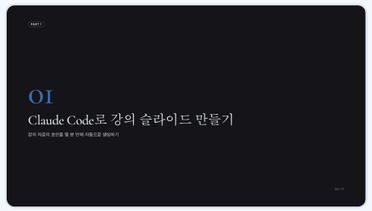
```
let rec fact product counter maxcounter =
  if counter > maxcounter then product
  else fact (product * counter) (counter + 1) maxcounter

let factorial n = fact 1 1 n
```

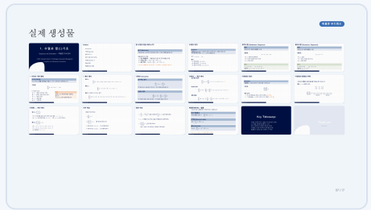
Higher-Order Functions

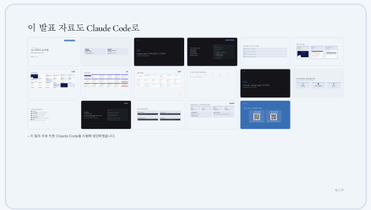
- Higher-order functions are functions that manipulate procedures; they take other functions or return functions as results.
- Higher-order functions provide a powerful tool for building abstractions and allow code reuse.

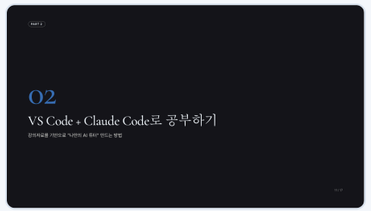
이 발표 자료도 Claude Code로

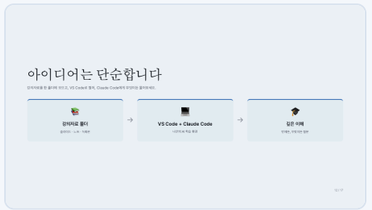


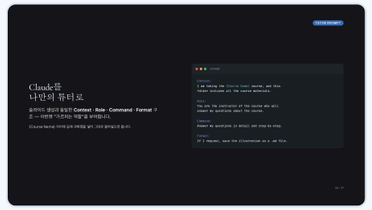


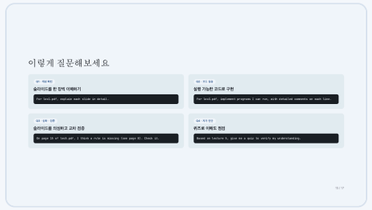












▸ 이 발표 자료 또한 Claude Code를 사용해 생성하였습니다.

PART 2

02

VS Code + Claude Code로 공부하기

강의자료를 기반으로 "나만의 AI 튜터" 만드는 방법

아이디어는 단순합니다

강의자료를 한 폴더에 모으고, VS Code로 열어, Claude Code에게 무엇이든 물어보세요.

강의자료 폴더

슬라이드 · 노트 · 녹화본

VS Code + Claude Code

나만의 AI 학습 환경

깊은 이해

언제든, 무엇이든 질문

5단계 학습 워크플로우

1. **과목 폴더 만들기**
   예: `ProgrammingLanguages` — 슬라이드·노트·과제 등 모든 자료를 넣습니다.

2. **VS Code로 폴더 열기**
   File → Open Folder, 그리고 사이드바의 Claude Code 패널을 엽니다.

3. **`/init` 으로 초기화**
   `CLAUDE.md` 가 생성되어 Claude가 폴더의 자료를 파악합니다.

4. **Claude를 튜터로 설정**
   Context·Role·Command·Format 프롬프트로 "이 과목의 교수자" 역할을 부여합니다.

5. **질문 시작!**
   슬라이드·다이어그램을 읽고 자세히 설명해 줍니다. 후속 질문을 이어가세요.

TUTOR PROMPT

Claude를 나만의 튜터로

슬라이드 생성과 동일한 **Context · Role · Command · Format** 구조 — 이번엔 "가르치는 역할"을 부여합니다.

{Course Name} 자리에 실제 과목명을 넣어 그대로 붙여넣으면 됩니다.

prompt

```
Context:
I am taking the {Course Name} course, and this
folder includes all the course materials.

Role:
You are the instructor of the course who will
answer my questions about the course.

Command:
Answer my questions in detail and step-by-step.

Format:
If I request, save the illustration as a .md file.
```

이렇게 질문해보세요

Q1 · 개념 확인

슬라이드를 한 장씩 이해하기

```
For lec1.pdf, explain each slide in detail.
```

Q2 · 코드 실습

실행 가능한 코드로 구현

```
For lec3.pdf, implement programs I can run, with detailed comments on each line.
```

Q3 · 심화 · 검증

슬라이드를 의심하고 교차 검증

```
On page 15 of lec5.pdf, I think a rule is missing (see page 8). Check it.
```

Q4 · 자가 진단

퀴즈로 이해도 점검

```
Based on lecture 5, give me a quiz to verify my understanding.
```

학습자 설문 · 응답 26명

학생들의 반응 — AI 학습법 설문 결과

4.6/5

전반적 만족도

96%가 4점 이상 (5점 만점)

9.2/10

추천 의향

81%가 9–10점으로 응답

88%

타 전공 과목 도입 희망

알고리즘·운영체제 등으로 확대

77%

주 3회 이상 활용

제공된 Claude Pro 구독 기준

가장 많이 활용한 학습 방식

- 강의 슬라이드·노트의 개념 설명 요청 (대다수)
- 과제·예제 코드 디버깅
- AI 생성 퀴즈로 자기 점검 · 시험 복습
- 타입 추론·의미론 등 심화 개념 질문

현장의 목소리 & 개선 제안

"정말 좋은 경험이었고, 이번 체험으로 개인적으로 Claude를 구독하게 되었습니다."

- 개선 요청: 구독 기간(1개월)·사용 한도 확대, 타 LLM(Codex·Gemini 등) 비교 소개, 과제 난이도 상향

GET STARTED

지금 바로 시작해보세요

1 강의자료 자동 생성

dgistpl.github.io/courses/claude_code/2026/

2 AI를 튜터로 활용하기

dgistpl.github.io/courses/ai_study/2026/