

# AI를 활용한 수업자료 자동 생성 프레임워크

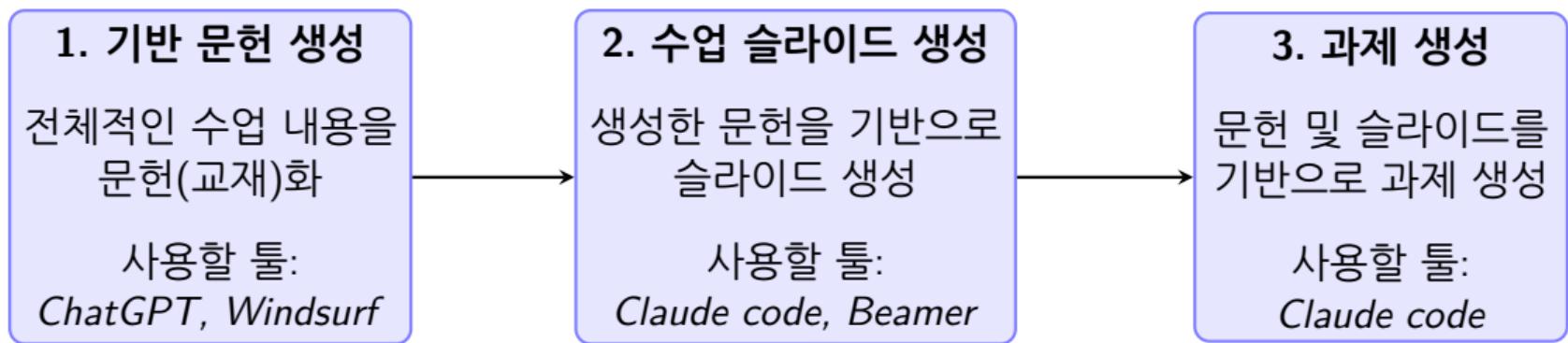
## 강의 슬라이드 및 과제 생성 자동화

전민석  
DGIST

2025년 11월 4일

# 수행방법: 3단계 프로세스

---



## 1단계: 기반 문헌 생성

---

# 1단계: 기반 문헌 생성

## 목표

수업에서 배울 내용들을 정리한 문헌을 생성

### 사용 도구

#### (1) ChatGPT

- 대화형 인공지능
- 글쓰기, 아이디어 발상

#### (2) Windsurf

- AI 기반 개발 환경
- 코드 작성, 실행, 디버깅
- 마크다운(.md) 형태로 문헌 생성

### 핵심 프로세스

1. ChatGPT로 로드맵 생성
2. Windsurf에서 구체적인 내용 작성
3. 단원별 학습 포인트 정리
4. 상세 내용 생성

## 1.1 ChatGPT를 사용한 로드맵 생성

---

예시: 자료구조 수업

ChatGPT에 프롬프트를 입력하여 수업의 전체 로드맵을 마크다운(.md) 형식으로 생성

사용한 프롬프트

*"I want to learn what data structures are and how we can use data structures. Would you please create a Learning Roadmap for me? Please describe the Roadmap using markdown format."*

# 1.1 ChatGPT를 사용한 로드맵 생성

- 결과물 (Roadmap.md): 전체 수업 구조, 단원별 주제, 학습 목표, 학습 순서

```
# 📚 Data Structures Learning Roadmap

## 1. **Foundations**
Before diving into data structures, build a strong foundation.

- **Prerequisites**
  - Learn a programming language (C, C++, Java, or Python recommended).
  - Understand basic programming concepts:
    - Variables, loops, conditionals
    - Functions & recursion
    - Arrays and strings
    - Memory concepts (stack vs heap)

 *Resources*:
  - *Introduction to Programming in Python* or *C Programming Absolute Beginner's Guide*
  - Online coding platforms (LeetCode, HackerRank)
```

## 1.2 Windsurf에서 구체적인 내용 생성

### Step 1: 로드맵 불러오기

ChatGPT가 생성한 Roadmap.md를 Windsurf에서 불러옴

```
## Roadmap.md > # Data Structures Learning Roadmap
1  # Data Structures Learning Roadmap
2
3  ## 1. **Foundations**
4  Before diving into data structures, build a
5  strong foundation.
6
7  - **Prerequisites**
8  - Learn a programming language (C, C++, Java, or Python recommended).
9  - Understand basic programming concepts:
10    - Variables, loops, conditionals
11    - Functions & recursion
12    - Arrays and strings
13    - Memory concepts (stack vs heap)
14
15  - *Resources*:
16  - *Introduction to Programming in Python* or
17  - *C Programming Absolute Beginner's Guide*
18  - Online coding platforms (LeetCode, HackerRank)
19
20  Files:
21  - [[01-Foundations.md]]
22
```

## 1.2 Windsurf에서 구체적인 내용 생성

---

### Step 2: 단원별 파일 생성

AI 채팅에 프롬프트를 입력하여 단원별로 배워야 할 포인트들이 작성된 파일들을 생성

#### 사용한 프롬프트 1

*"This is a learning roadmap report. I need you to do the following:*

*Step 1. Based on the learning roadmap, create a list of markdown files for each topic.*

*Step 2. For each topic, write an introduction and list of knowledge points. Each knowledge point title should be written in [ ]"*

#### 사용한 프롬프트 2

*"I need you to explain the knowledge points in detail"*

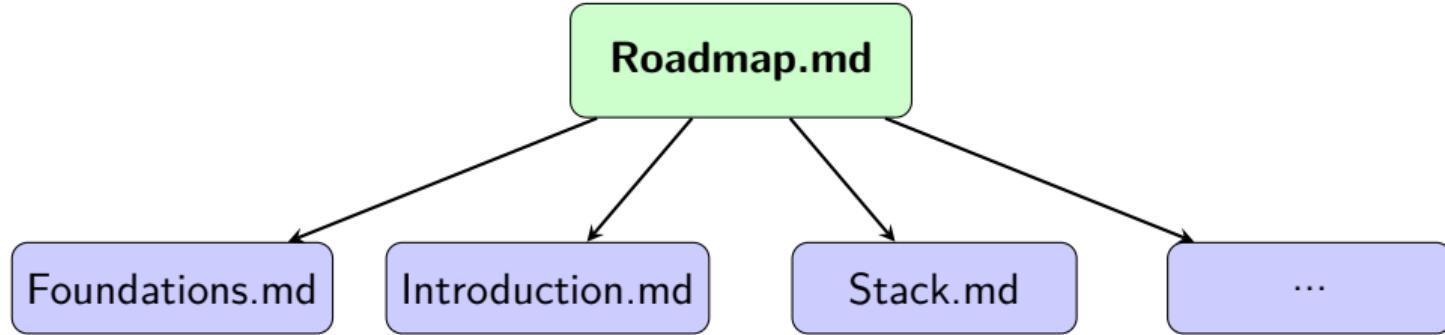
# 1.2 Windsurf에서 구체적인 내용 생성

## 최종 결과

단원별로 체계적으로 정리된 기반 문헌 파일들이 완성됨

```
topics > Stacks.md > # Stacks > 
1 # Stacks
2
3 LIFO structure for managing nested operations and histories.
4
5 ## Knowledge Points
6 - [[Core operations: push/pop/peek]]
7 - [[Array vs linked-list implementation]]
8 - [[Expression evaluation and parentheses]]
9 - [[Function call stack and recursion]]
10 - [[Undo/redo and backtracking]]
11 - [[Time/space complexity]]
12
13 ## Details
14
15 ### Core operations: push/pop/peek
16 - Push: add an element to the top.
17 - Pop: remove and return the top element. Underflow if empty.
18 - Peek: return the top element without removing it.
19 - Empty/Size: check if the stack is empty and how many elements it has.
20
21 Example (Python list as stack):
22 ````python
23 s = []
```

# 기반 문헌 생성 결과



## 디렉토리 구조

```
DataStructure/
|- Roadmap.md
|- topics/
  |- Foundations.md
  |- ...
```

## 2단계: 슬라이드 생성

---

# 2단계: 슬라이드 생성

## 목표

생성한 문헌을 기반으로 수업 슬라이드 생성하기

### 사용 도구

#### Beamer

- LaTeX 기반 문서 클래스
- 학술 및 전문 발표용 슬라이드 제작
- 코드로 슬라이드를 생성

#### Claude Code

- Claude AI 기반 코딩 보조 도구
- 코드 작성, 수정, 설명, 디버깅 지원

### 작업 순서

1. slides/ 폴더 생성
2. Beamer 템플릿 선택
3. Claude Code 실행
4. /init 명령어로 코드베이스 이해
5. 슬라이드 생성 프롬프트 입력

## 2.1 슬라이드 생성을 위한 세팅

---

### 디렉토리 구조 설정

```
DataStructure/  
| - Roadmap.md  
| - topics/  
|   | - Foundations.md  
|   | - ...  
| - slides/  
|   | - template/
```

### Claude Code 초기화

1. Root 폴더(DataStructure/)에서 Claude Code 실행
2. /init 명령어 실행

## 2.2 Claude Code로 슬라이드 생성

---

### 프롬프트

Currently, I am teaching Data Structure course based on the markdown files in the 'topics/' folder.

In the lecture, I will use slides that are constructed based on the markdown files in the 'topic/' folder.

I will use Beamer to generate slides, and the template I will use is placed in the 'slides/BeamerTemplate/' folder.

I need you to generate a folder 'slides/Stack/' and write a Beamer LaTeX file stack.tex based on the contents in 'topics/Stack.md' and referring to the template in 'slides/BeamerTemplate'.

## 2.2 Claude Code로 슬라이드 생성

### 결과

### AI에 의해 자동으로 생성된 전문적인 강의 슬라이드

Data Structures: Stacks

Data Structure Course  
Version 2.001  
Updated 3/2023

Contents

- 1. Introduction to Stacks
- 2. Core Operations
- 3. Implementation Approaches
- 4. Applications
- 5. Complexity Analysis
- 6. Summary

Introduction to Stacks

What is a Stack?

Definition

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle.

Key characteristics

- Elements are added and removed from the same end (top)
- Only the top element is accessible
- Perfect for managing nested operations and recursive functions
- Natural structure for function calls and recursion

Stack Structure

Core Operations

Essential Stack Operations

Initial Content: [ ]

Example:

- + Push: Add elements to top
- + Pop: Remove and return top element
- + Peek/Top: Return top element
- + Empty: Check if stack is empty
- + Size: Get number of elements

User Comments: All operations are O(1) - constant time (unrelated to push in dynamic arrays)

Stack Operations Visualization

Initial Stack: [ ]

After Push(10): [10]

Push: 10

After Pop(): [ ]

Pop: 10 = TEP

Implementation Approaches

Array-based vs Linked List Implementation

Array-based Stack	Linked List-based Stack
Pros:	Pros:
- Simple memory	- Easy to implement
- Great cache locality	- Dynamic size
- Simple implementation	- Good for pointer manipulation
- O(1) amortized cost	- Extra pointer memory
- Occupies more space	- Poor cache locality
- Capacity management	- More complex

Linked List Stack Implementation

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

class Stack:
    def __init__(self):
        self.head = None
        self.size = 0

    def push(self, val):
        new_node = Node(val)
        new_node.next = self.head
        self.head = new_node
        self.size += 1

    def pop(self):
        if self.size == 0:
            return None
        else:
            popped_val = self.head.val
            self.head = self.head.next
            self.size -= 1
            return popped_val
```

Applications

Expression Evaluation: Parenthesis Matching

Check if parentheses, brackets, and braces are properly balanced.

Algorithm:

1. Push opening brackets onto stack
2. If closing bracket is encountered:
  - Check if it's empty
  - Check if top matching opening bracket exists
  - If yes, pop the opening bracket off the stack
  - If no, then stack is unbalanced
3. Stack should be empty at end

Implementation Note: Use stack overflow for deep recursion.

Function Call Stack and Recursion

Call Stack Mechanism

- Each function call creates a stack frame
- Frame contains: parameters, local variables, stack pointer, and return address
- Because each call stack implicitly contains previous stack frames, deep recursion can cause stack overflow.

Call Stack

Undo/Redo Operations

Implementation Note: Undo/redo actions performed on actions

- Undo/redo
- Undo/redo actions
- Actions push to undo, clear redo
- Undo/redo stack takes O(n) time
- Redo: push to redo, apply redo
- Undo: push to undo, apply undo to redo

Implementation Note: Undo/redo stack is used stack overflow for deep recursion.

Complexity Analysis

Time and Space Complexity

Implementation	Push	Pop/Peek	Space
Array-based	O(1)	O(1)	O(n)
Linked List-based	O(1)	O(1)	O(n) + pointer overhead

Implementation Note: Array-based O(1) push due to reuse. Linked list-based O(n) because uses single push. O(n) pointer overhead.

Complexity Analysis

- Amortized O(1) push due to reuse
- Linked list-based: amortized O(1)
- Space: O(n) + pointer overhead

Summary

Key Takeaways

Implementation

- LIFO data structure with top-only access
- Essential operations: push, pop, peek, empty, size
- 30 operations are O(1) or linear complexity

Implementation Approaches

- Array-based: better cache, amortized O(1)
- Linked list-based: guaranteed O(1), more memory

Implementation Examples

- Expression evaluation and parentheses matching
- Function argument and parameter passing
- Stack-based functionality

Thank You!

Questions?

## 3단계: 과제 생성

---

# 3단계: 과제 생성

## 목표

생성한 문헌을 기반으로 과제 생성하기

### 디렉토리 구조

DataStructure/

| - Roadmap.md

| - topics/

| - slides/

| - assignments/

### 생성 과정

1. assignments/ 디렉토리 생성

2. Claude Code에 프롬프트 입력

3. 문헌 및 슬라이드 기반 과제 생성

## 3단계: 과제 생성

---

### 프롬프트

Currently, I am teaching Data Structure course based on the markdown files in the 'topics/' folder.

In the lecture, I will give the students assignments about implementing each data structure.

I need you to generate 'assignments/stack' folder and implement 'stack.py' that implements the core functions of stack data structure, and also write an application code 'application.py' that imports and uses the functions in stack.py.

After you implement the core functions in stack.py, I will hide them and make the student implement them as their assignments.

# 3단계: 과제 생성

## 결과

### 자동으로 생성된 체계적인 과제

README.md	update	2 weeks ago
application.py	update	2 weeks ago
stack.py	update	2 weeks ago
test_stack.py	update	2 weeks ago

README.md



## Stack Assignment

This assignment contains a complete implementation of stack data structures and their applications.

```
class ArrayStack:  
    """  
    Array-based stack implementation using Python list.  
  
    Provides O(1) amortized time complexity for all operations.  
    """  
  
    def __init__(self):  
        """Initialize an empty stack."""  
        self._data = []  
  
    def push(self, item):  
        """  
        Add an item to the top of the stack.  
  
        Args:  
            item: The item to be added to the stack  
  
        Time Complexity: O(1) amortized  
        """  
        self._data.append(item)
```

# 결론

---

## AI 기반 수업자료 생성 시스템의 핵심

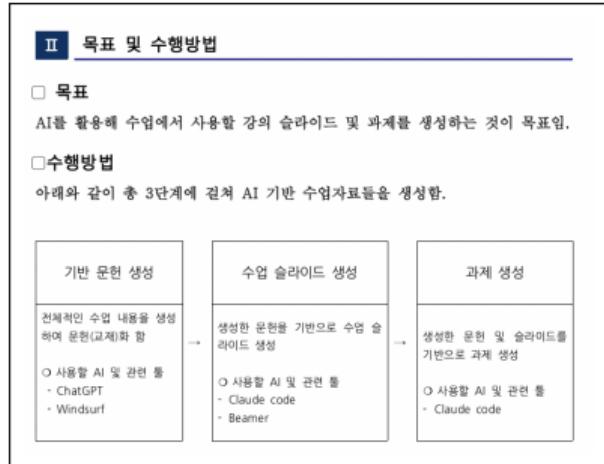
1. **기반 문헌 생성**: ChatGPT + Windsurf로 체계적인 교재 작성
2. **슬라이드 생성**: Claude Code + Beamer로 전문적인 프레젠테이션 제작
3. **과제 생성**: AI 기반 맞춤형 과제 자동 생성

## 핵심 가치

효율성 × 품질 × 확장성

# 확장성: 발표 슬라이드 생성

- 제출한 아이디어 결과서로부터 발표 자료 생성하는 것 또한 가능



⇒  
Claude code  
+  
Windsurf

AI를 활용한 수업자료 자동 생성 프레임워크  
강의 슬라이드 및 과제 생성 자동화

전민석  
DGIST

2025년 11월 4일

제출한 아이디어 결과서

오늘 발표에 사용한 슬라이드

# 감사합니다!

생성한 코드: <https://github.com/MinseokJGit/DataStructure/>

생성한 과목 페이지: [https://dgistpl.github.io/courses/ai\\_ds/2025/](https://dgistpl.github.io/courses/ai_ds/2025/)