생성형 AI 소프트웨어공학 · 2026학년도 2학기 · 3강

생성형 AI의 이해와 지시문 작성 기초

왜 같은 지시에 다른 답이 나오는가: 원리 · 네 기둥 구조 · 비교 실험, 그리고 과제 1

대전대학교 컴퓨터공학과 · 박상돈

3강 · 수업 안내

지난 시간, 그리고 오늘

지난 시간 요약

정답 프로세스 모델은 없고, 상황이 모델을 고릅니다.

AI 시대의 개발은 지시 → 생성 → 확인 → 수정의 초소형 스프린트 반복입니다.

그리고 여러분 노트북에는 이제 AI 에이전트 둘, Codex(내 ChatGPT 계정)와 ZCode(수업 공용 계정)가 연결되어 있습니다. 오늘은 그 AI를 '잘 부리는 말'을 배웁니다.

이번 주 성취 기준

- 생성형 AI가 답을 만드는 원리와 한계(비결정성 · 환각 · 지식 경계)를 설명할 수 있다.
- 역할 · 맥락 · 요구 · 형식을 갖춘 지시문을 작성할 수 있다.
- 같은 기능을 다른 지시문으로 요청하고, 결과 차이를 근거와 함께 분석할 수 있다.

01

PART 1

생성형 AI의 이해

부리려면 먼저 알아야 합니다. AI가 답을 '만들어 내는' 원리와 세 가지 한계를 봅니다.

생성형 AI는 실제로 무엇을 하는가

대규모 언어 모델(LLM)은 지금까지의 글을 읽고, **다음에 올 조각(토큰)을 확률로 고릅니다**.
대화, 번역, 코드 생성. '이해하는 것처럼' 보이는 모든 일이 이 한 가지 동작을 되풀이한 결과입니다.

① 읽는다

지금까지의 글 전체. 여러분의 지시문과 이전 대화가 모두 입력입니다.

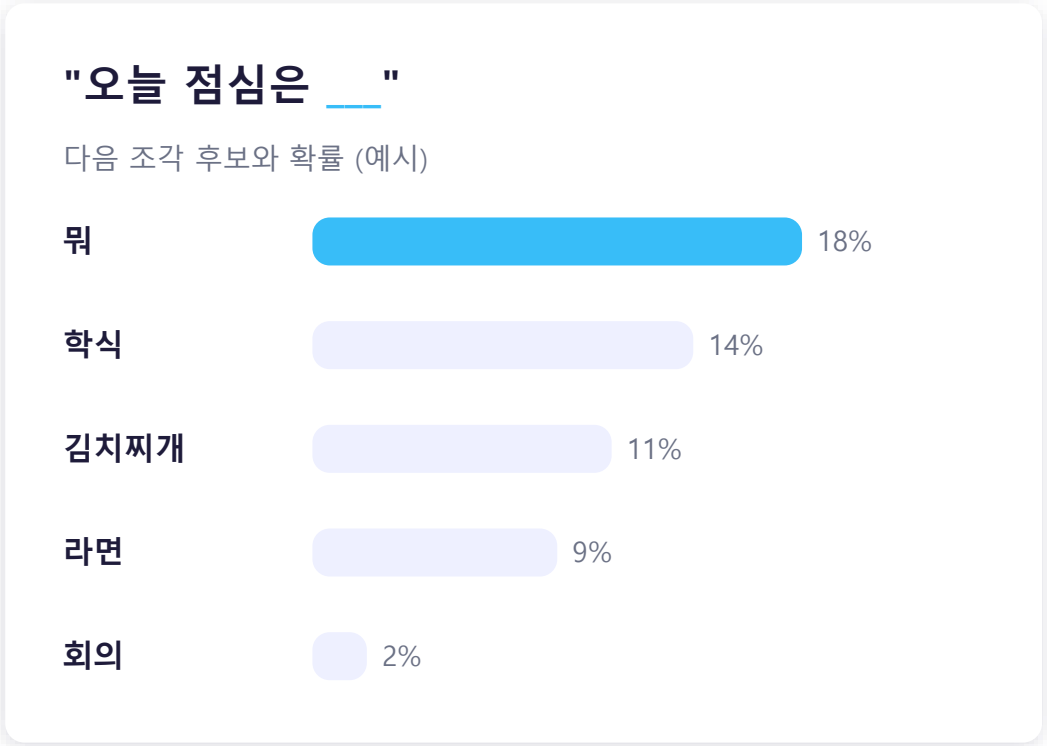
② 계산한다

다음에 올 수 있는 모든 조각에 대해 "올 법한 정도"를 확률로 매깁니다.

③ 뽑아 붙이고, 반복한다

확률에 따라 하나를 뽑아 덧붙이고, 다시 ①로. 답이 끝날 때까지 반복합니다.

다음 조각 맞히기, 눈으로 보기



모델은 이 후보들 중 하나를 확률에 따라 '뽑아' 문장에 붙입니다. 그리고 붙은 결과를 다시 읽고, 또 다음 조각을 예측합니다. 답이 완성될 때까지.

코드도 똑같습니다. "def c_to_f(c):" 다음에는 들여쓰기와 return이 올 확률이 압도적으로 높죠. 지난주 temp.py가 그렇게 만들어진 겁니다.

토큰(token) = 단어보다 작은 글 조각(한글은 음절·부분 단위로 쪼개짐). 이 수업에선 "조각" 정도의 이해면 충분합니다.

왜 같은 지시에 다른 답이 나오는가

확률에서 '뽑기' 때문입니다. 매번 주사위를 굴리는 셈이라, 같은 입력에도 다른 경로의 답이 나옵니다. 뽑기의 과감함을 조절하는 다이얼이 온도(temperature)입니다.

온도 낮음: 안전하게

가장 확률 높은 후보 위주로 선택합니다. 답이 비슷하게 반복됩니다. 정형적인 코드 생성에 유리합니다.

온도 높음: 과감하게

낮은 확률의 후보도 종종 선택합니다. 답이 다양하고 창의적이지만, 예측하기 어려워집니다. 아이디어 발산에 유리합니다.

공학적 결론: 답이 달라지는 것은 오류가 아니라 원래 성질입니다
결과가 매번 달라질 수 있다면, 결과는 매번 확인해야 합니다. 1강에서 "검증은 사람의 몫"이라고 한 이유가 여기 있습니다.

환각(Hallucination): 그럴듯한 거짓말

모델이 잘하도록 훈련된 것은 "그럴듯한 다음 조각"이지 "사실"이 아닙니다. 그래서 모르는 것을 물어도 자신 있는 말투로, 그럴듯하게 지어냅니다.

코딩에서 가장 흔한 형태가 이것입니다. 있을 법하지만 존재하지 않는 함수.

환각의 전형 (예시)

```python
import pandas as pd

df = pd.read_excel("성적.xlsx")
df.auto_grade("총점")
# ↑ 그럴듯하지만, 존재하지 않는 함수
#   실행하면 AttributeError
```

말투로는 진위를 구분할 수 없다
맞을 때도, 틀릴 때도 같은 확신의 문장으로 말합니다. "자신 있어 보임"은 판단 근거가 아닙니다.

판별법은 두 가지뿐
실행해 본다(오류가 즉시 알려 줌) · 공식 문서를 찾아본다. 함수 이름, 옵션, 사실 관계는 이 둘로만 확인합니다.

모델이 모르는 것: 지식의 두 경계

시점의 경계
학습 데이터가 끝난 시점 이후의 세계를 모릅니다. 어제 나온 라이브러리 버전, 최신 문법 변화, 오늘의 뉴스.

접근의 경계
공개된 적 없는 정보를 모릅니다. 우리 수업의 규칙, 여러분 과제의 요구사항, 우리 팀 저장소의 코드.

그래서, 모르는 것은 내가 준다
경계 밖의 정보는 지어내기(환각) 전에 지시문에 '맥락'으로 담아 줍니다. 2부에서 배울 기둥 하나가 바로 이것입니다.

흔한 착각
"AI니까 우리 과제 내용도 알겠지"? 모릅니다. 여러분이 말해 주지 않은 것은 존재하지 않는 정보이고, 그 빈칸에서 환각이 태어납니다.

대화의 착시: 매번 처음부터 다시 읽습니다

1번째 답변 전	지시 ①				
2번째 답변 전	지시 ①	답 ①	지시 ②		
3번째 답변 전	지시 ①	답 ①	지시 ②	답 ②	지시 ③

→ 모델이 '읽는' 범위. 기억이 아니라 매번 전체를 다시 읽는 것

읽는 창의 크기는 유한합니다 (컨텍스트 창)

대화가 아주 길어지면 앞부분이 밀려나 잊힌 것처럼 됩니다. 오래된 제약이 슬그머니 무시되는 이유.

실전 습관 두 가지

새 작업은 새 대화로 시작하고, 중요한 제약은 다시 말해 줍니다. 오늘 실습에서 지시문마다 새 대화를 여는 이유입니다.

원리에서 습관으로

확률로 말한다 (비결정성)
→ 결과는 실행과 기준으로 '판정'합니다. 한 번 잘 나왔다고 다음에도 그러리라 믿지 않습니다.

그럴듯하게 지어낸다 (환각)
→ 함수 이름 · API · 사실 관계는 실행과 공식 문서로만 확인합니다. 말투는 근거가 아닙니다.

모르는 것이 많다 (지식의 경계)
→ 시점 밖 · 비공개 정보는 내가 지시문에 맥락으로 담아 줍니다.

이 세 습관을 문장에 담는 틀이 있습니다
역할 · 맥락 · 요구 · 형식, 네 기둥의 지시문 구조. 지금부터 2부에서 하나씩 세웁니다.

02

PART 2

지시문 작성 기초

내가 비운 칸은 AI가 그럴듯하게 채웁니다. 빈칸을 채우는 네 기둥, 역할 · 맥락 · 요구 · 형식.

"계산기 만들어 줘", 무엇이 문제인가

"계산기 만들어 줘."

틀린 지시는 아닙니다. 다만 아래 결정들을 전부 AI에게 넘긴 지시입니다.

어떤 계산기?
사칙연산? 공학용? 환율 계산?

어떤 모양?
터미널 입력? 창(GUI)? 웹 페이지?

이름은?
파일 이름과 함수 이름은 무엇으로?

예외는?
0으로 나누기, 숫자 아닌 입력은?

내가 비운 칸은 AI가 그럴듯하게 채웁니다
그리고 그 결정의 책임은 제출한 사람, 즉 나에게 남습니다(1강 규칙 ②). 좋은 지시문 = 중요한 빈칸을 내가 먼저 채운 지시문.

지시문의 네 기둥

역할 (Role)

AI가 '누구로서' 답할지. 답변의 관점과 눈높이를 정합니다. "파이썬 입문 수업의 조교로서"

맥락 (Context)

AI가 알 수 없는 상황 · 제약 · 자료. 지식의 경계를 내가 메웁니다. "표준 라이브러리만, 우리 저장소에 추가"

요구 (Task)

무엇을 할지. 통과·불통과를 판정할 수 있는 구체적 동사와 결과로. "...를 반환하는 함수를 작성"

형식 (Format)

결과물의 모양. 파일·함수 이름, 주석 언어, 확인용 출력까지 지정합니다. "gradeC.py에, 한국어 주석으로"

기둥 ① 역할: 누구로서 답할 것인가

역할은 답의 '정답'보다 답의 관점 · 눈높이 · 말투를 바꿉니다.

역할 없음

"정규표현식 설명해 줘."

누구에게 설명하는지 정보가 없으니, 숙련자 기준의 압축된 설명이 나오기 쉽습니다. 기호가 쏟아지고, 처음 보는 사람은 세 줄 만에 길을 잃습니다.

역할 지정

"파이썬 입문 수업의 조교로서, 정규표현식을 처음 보는 1학년에게 설명해 줘."

쉬운 예시부터 단계적으로, 전문용어에는 풀이가 붙습니다. 같은 주제, 같은 모델인데 눈높이가 달라집니다.

우리 수업 기본 역할 추천: "파이썬 입문 수업의 조교로서, 1학년이 읽을 수 있게". 코드가 쉬워지고 설명이 붙어, '내가 설명할 수 있는 코드'를 받기 좋아집니다.

기둥 ② 맥락: AI가 모르는 것을 내가 준다

어디에 쓰이는지 · 무엇이 금지인지 · 어떤 입력이 들어오는지. 지식의 경계 밖 정보를 채웁니다.

맥락 없음

"이메일 형식 검사 함수 만들어 줘."

AI가 임의로 채우는 것들: 외부 검증 라이브러리를 설치해 쓰거나, 빈 문자열 입력을 고려하지 않거나, 우리 프로젝트와 무관한 구조로 만들 수 있습니다.

맥락 지정

"회원가입 폼의 입력 검사용이야. 파이썬 3.13 표준 라이브러리만 사용하고, 빈 문자열도 들어올 수 있어."

쓰임새 · 금지 사항 · 입력 조건이 정해지니, 결과가 우리 상황에 바로 맞습니다. 환각이 자랄 빈칸이 줄어듭니다.

맥락 점검 질문: "내 머릿속에만 있고 지시문에는 없는 전제가 무엇인가?" 그 전제가 곧 써야 할 맥락입니다.

기둥 ③ 요구: 판정할 수 있게 말한다

좋은 요구의 기준은 하나. 결과를 받았을 때 통과 · 불통과를 '판정'할 수 있는가.

판정 불가능한 요구

"코드 좀 잘 정리해 줘."

'잘'의 기준이 없으니 결과를 받아도 됐는지 판단할 수 없습니다. 지시한 나도, 만든 AI도, 서로 다른 '잘'을 생각하고 있습니다.

판정 가능한 요구

"함수 이름을 동사형으로 바꾸고, 20줄이 넘는 함수는 두 개 이상으로 분리해 줘."

받자마자 확인할 수 있습니다. 동사형인가? 20줄 넘는 함수가 남았는가? 판정 기준이 지시문 안에 들어 있습니다.

지난주 첫 지시를 떠올려 보세요. "25도를 변환한 결과를 출력해"가 있었기에 77.0으로 판정할 수 있었습니다. 요구에는 확인 방법이 함께 담깁니다.

기둥 ④ 형식: 결과물의 모양을 지정한다

내용이 맞아도 모양이 제각각이면 팀에서는 못 씁니다. 이름 · 언어 · 확인 코드까지 지정합니다.

형식 없음

(요구만 쓰고 형식 생략)

파일 이름은 AI 마음대로(validator.py? main.py?), 주석은 영어로, 확인 코드는 없이. 받을 때마다 모양이 달라서 저장소가 뒤죽박죽이 됩니다.

형식 지정

"check_email.py 파일에, 함수 이름은 is_valid_email, 주석은 한국어로. 파일 끝에 사용 예 3개를 출력하는 코드를 붙여 줘."

받는 순간 자리(파일) · 부를 이름(함수) · 읽을 언어(주석) · 확인 방법(사용 예)이 전부 정해져 있습니다.

형식은 취향 문제가 아니라 협업 문제입니다. 6주차 설계, 7주차 명세 기반 구현에서 형식 지정이 팀의 생명선이 됩니다.

조립: 네 기둥이 모두 선 지시문

역할 파이썬 입문 수업의 조교로서, 1학년이 읽을 수 있게 작성해 줘.

맥락 회원가입 폼의 입력 검사에 쓸 거야. 파이썬 3.13 표준 라이브러리만 사용하고, 빈 문자열도 들어올 수 있어.

요구 문자열이 이메일 형식이면 True, 아니면 False를 반환하는 함수를 작성해 줘. 빈 문자열은 False야.

형식 check_email.py 파일에, 함수 이름은 is_valid_email, 주석은 한국어로. 파일 끝에 사용 예 3개를 출력해 줘.

이 구조로 쓰면 AI가 임의로 정할 빈칸이 거의 없습니다. 문단 하나로 이어 써도, 이렇게 줄을 나눠 써도 좋습니다.

지시문도 반복으로 다듬습니다

첫 지시에 완벽할 필요는 없습니다
작은 작업은 요구 + 형식 정도로 가볍게 시작해도 됩니다. 네 기둥은 의무가 아니라 점검표입니다.

결과와의 '차이'를 말로 정리해 재지시
2강의 루프 그대로입니다. "빈 문자열에서 오류가 나. 빈 문자열은 False를 반환하게 고쳐 줘." 차이가 곧 다음 지시문입니다.

지시문을 버리지 말고 기록하세요
이 수업에서 지시문은 산출물입니다. 과제 1도, 팀 프로젝트도 '어떤 지시로 이 결과를 얻었는가'를 함께 제출하고 평가합니다.

그럼 정말 결과가 그렇게 달라질까?
말로는 충분합니다. 이제 같은 기능을 세 가지 지시문으로 시켜서, 차이를 데이터로 확인합니다. 3부 실습 갑니다.

03

PART 3

지시문 비교 실험

같은 기능, 세 가지 지시문. 결과가 얼마나 달라지는지 직접 데이터로 확인합니다. 오늘 실습이 곧 과제 1의 예행연습입니다.

실험 설계: 대상 기능과 규칙

실험 대상 기능: 점수 목록을 받아 학점(A~F) 목록으로 바꾸는 변환기

작지만 결정할 빈칸이 많아 실험에 딱 좋은 기능입니다.

경계값
90점은 A? 89.5는? 60은 D?

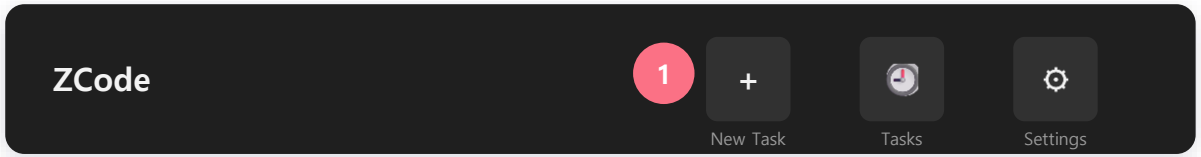
입력
정수만? 실수도? 범위 밖 값은?

이름
파일 이름 · 함수 이름은?

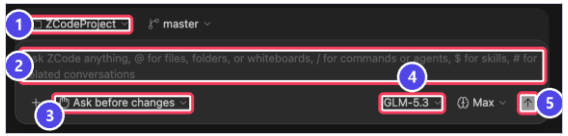
확인
무엇이 나오면 성공인가?

실험 규칙 세 가지
지시문마다 ZCode에서 'New Task'(새 작업)로 시작합니다(앞 대화가 다음 실험을 오염시키지 않도록. 1부 맥락 창). 결과는 각각 gradeA.py · gradeB.py · gradeC.py로 저장합니다. 세 실험 모두 같은 도구, 같은 모델(ZCode, GLM-5.3)로 돌립니다. 중간에 바꾸면 비교가 안 됩니다.

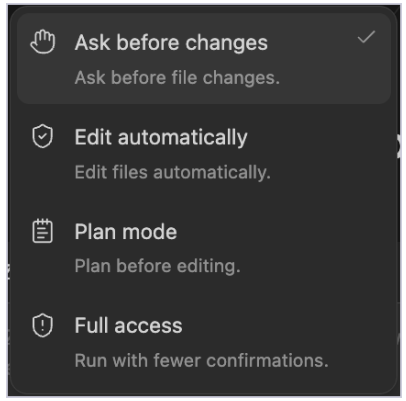
실험 진행 절차: 한 실험은 이렇게 돌립니다 (A · B · C 공통)



ZCode 창 위쪽 구성도. + (New Task)가 '새 대화'입니다. 실험 A·B·C 각각 시작 전에 반드시 누릅니다. Codex라면 패널의 '+ New chat'.



② 입력창에 지시문을 붙여넣고 보내기 (출처: ZCode 공식 문서)



③ 파일을 만들기 전에 'Ask before changes' 모드가 허락을 묻습니다. 읽고 Allow (출처: ZCode 공식 문서)

④ VS Code 터미널: 실행해서 확인

```
python gradeA.py        # 결과가 화면에 찍히면 실행 성공
```

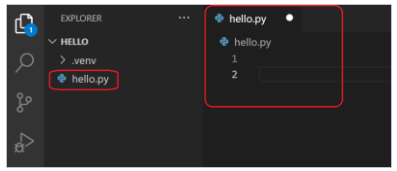
1 **+ New Task, 새 대화로 시작**
앞 실험의 대화가 남아 있으면 다음 실험이 오염됩니다 (1부 '매번 다시 읽기'). A·B·C 각각 새 대화.

2 **지시문을 '그대로' 붙여넣기 → Enter**
슬라이드의 지시문을 복사해 입력창에. 오타 하나가 결과를 바꿉니다.

3 **승인: 무조건 누르지 말고 읽고**
ZCode가 만들 파일 내용을 보여 주면 훑어보고 Allow. 명령 실행도 마찬가지. Codex도 같은 식으로 묻습니다.

4 **실행 → 결과 관찰**
터미널에서 python 파일명.py. 실행이 되는지, 무엇이 찍히는지 적어 둡니다.

5 **파일 이름 맞추기: gradeA/B/C.py**
AI가 다른 이름으로 만들었으면 탐색기에서 파일 클릭 → F2 → 이름 변경. C는 지시문에 이름이 있어 그대로 나옵니다.



탐색기(왼쪽)에서 파일을 클릭 후 F2 → 새 이름 입력 (출처: VS Code Docs)

3부 · 지시문 비교 실험

실험 A: 한 줄짜리 지시

지시문 A (새 대화에서 그대로 입력)

"학점 계산기 만들어 줘."

실행하고 결과를 열어, AI가 '대신 정해 버린 결정'을 다섯 개 이상 찾아 적으세요.

관찰 예상 ① 형태가 제각각

함수만 있는 사람, input()으로 묻는 사람, 반복문이 도는 사람. 옆 사람과 비교해 보세요.

관찰 예상 ② 경계와 이름이 임의

90의 학점, 파일·함수 이름이 사람마다 다를 겁니다. 내가 안 정했으니 AI가 뽑은 값입니다.

옆 사람과 결과가 다르다는 것 자체가 오늘의 데이터입니다. 그 차이를 기록하세요.

실험 B: 요구만 명확하게

지시문 B (반드시 '새 대화'에서 입력)

"0부터 100 사이 정수 점수의 리스트를 받아서, 90 이상은 A, 80 이상은 B, 70 이상은 C, 60 이상은 D, 그 미만은 F로 바꾼 리스트를 반환하는 파이썬 함수를 만들어 줘."

요구(기둥 ③)가 서니 기능의 뼈대는 같아집니다. 그러나 아직 AI가 정하는 것들:

이름
파일 · 함수 이름은 여전히 AI 마음대로

범위 밖 값
-5나 150이 오면? 지시에 없어 임의 처리

확인 방법
실행 예가 없어 판정을 내가 만들어야 함

실험 C: 네 기둥을 모두 세운 지시

역할 파이썬 입문 수업의 조교로서, 1학년이 읽을 수 있게 작성해 줘.

맥락 hello-sw 저장소에 추가할 단일 파일이고, 표준 라이브러리만 사용해.

요구 0~100 정수 점수 리스트를 받아 90↑ A, 80↑ B, 70↑ C, 60↑ D, 그 외 F의 리스트를 반환하는 함수. 범위 밖 값이 있으면 ValueError를 발생시켜.

형식 파일 gradeC.py, 함수 이름 to_grade, 주석은 한국어로. 파일 끝에 print(to_grade([95, 82, 61, 45])) 확인 코드를 포함해 줘.

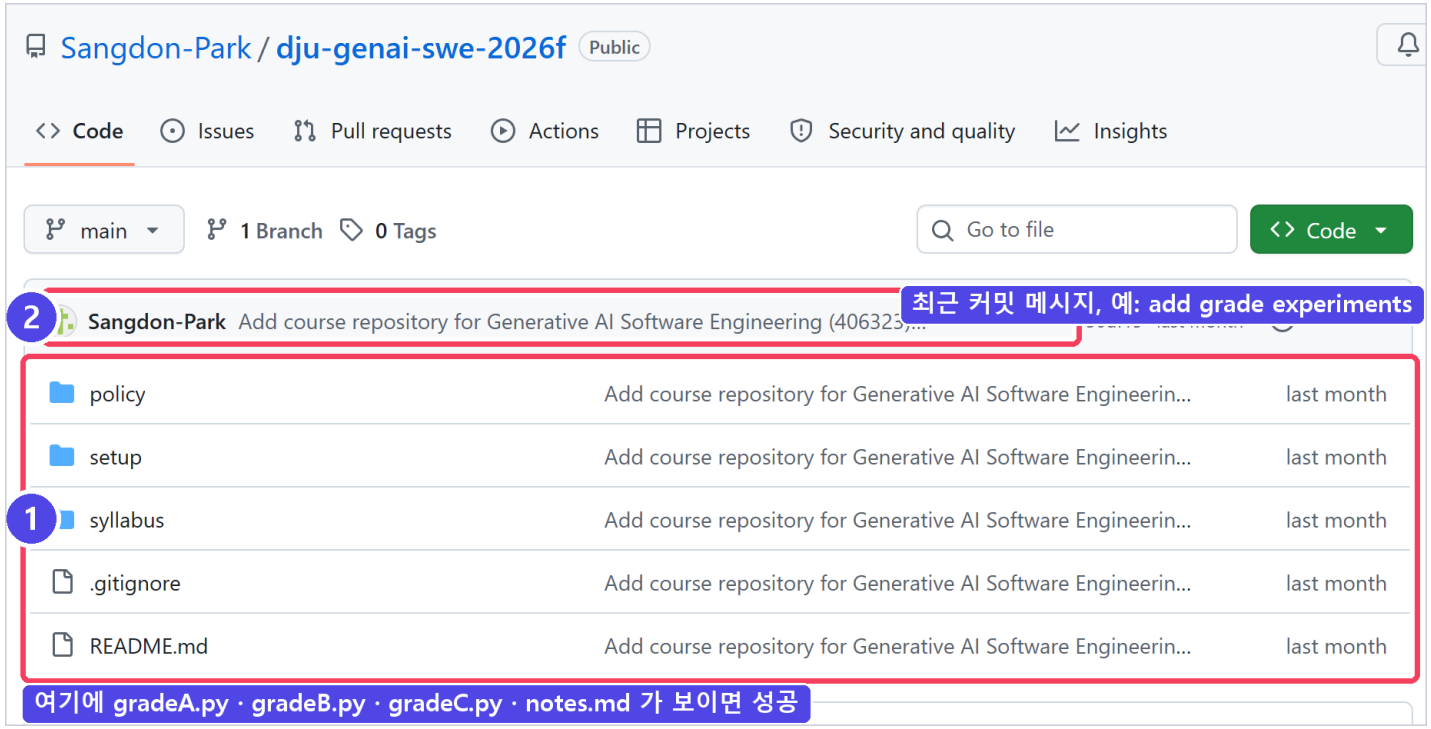
실행하면 ['A', 'B', 'D', 'F']. 판정 기준까지 지시문 안에 있으니, 받는 즉시 통과 여부를 알 수 있습니다.

결과 비교: 다섯 가지를 관찰하세요

관찰 항목	확인 질문 (A · B · C 각각에 답해 보세요)
경계값	90점은 무엇이 되었나? 세 버전의 답이 서로 같은가?
입력 방어	-5나 150을 넣으면 각각 어떻게 되나? (오류 · 무시 · 엉뚱한 학점?)
이름	파일 · 함수 이름을 누가 정했나? 나인가, AI인가?
실행	받은 코드가 고치지 않고 한 번에 실행되었나?
설명 가능성	코드의 각 줄을 내 말로 설명할 수 있나? (1강 규칙 ②)

차이 3가지 이상을 '문장'으로 기록
"A는 경계 89를 B로 처리했지만 C는 ValueError를 냈다"처럼 코드 근거와 함께. 이 기록이 그대로 과제 1의 분석 문단이 됩니다.

세 파일과 관찰 기록을 저장소에 올리기 (push)



push 후 브라우저에서 hello-sw 저장소를 새로고침(F5). 파일 목록에 네 파일이 보이면 성공 (예시는 제 강의 저장소 화면)

1 **notes.md 만들기**

VS Code 탐색기 → 새 파일 → notes.md. 아까 적은 차이 3가지 이상을 문장으로 붙여넣고 저장. (.md = 메모용 텍스트 파일)

VS Code 터미널 (hello-sw 안)

```
git add .          # . = 바뀐 파일 전부
git commit -m "add grade experiments"
git push
```

2 **add . → commit → push**

지난주와 같은 세 단계. 이번엔 add 뒤에 점(.)을 붙여 새로 만든 파일 전부를 한 번에 담습니다.

push 전 마지막 점검: 파일 안에 키나 비밀번호가 없는지

AI가 만든 코드에 비밀이 들어갈 일은 없지만, 2강 규칙이니 습관처럼 확인합니다 (Ctrl+Shift+F 로 key, password, token 검색 → 의심스러운 게 없으면 통과). 올린 뒤 저장소 주소를 LMS 3주차 게시판에 제출하면 오늘 출석 인증 완료.

공지 · 과제 1

과제 1: 지시문 비교 실험 보고서

대상: 오늘과 '다른' 작은 기능 하나

스스로 선택합니다(예: 온도 통계, 단어 세기, 비밀번호 규칙 검사). 결정할 빈칸이 있는 작은 기능.

방법: 오늘 실습과 동일

지시문 A(한 줄) · B(요구만) · C(네 기둥)를 각각 새 대화에서 실행하고 결과를 저장합니다.

분석: 1쪽 (analysis.md)

차이 3가지 이상을 코드 근거와 함께. 어떤 지시문의 어떤 문장이 그 차이를 만들었는지.

제출 · 마감 · 배점

제출물: 지시문 3개 · 생성 코드 3개 · analysis.md, 전부 개인 저장소에 push 후 주소를 LMS 과제 1에 제출

마감: 4주차 수업 시작 전

배점: 과제 10% 중 1회차

평가 포인트

네 기둥 구조의 반영 · 분석의 구체성(코드 근거 인용) · 설명 가능성. 제출물에 대해 구두로 확인할 수 있습니다.

실습 체크리스트: 나가기 전에

- [x] gradeA.py · gradeB.py · gradeC.py 를 만들어 모두 실행해 보았다.
- [x] 지시문마다 '새 대화'에서 실행했다.
- [x] 다섯 관찰 항목으로 비교하고, 차이를 3가지 이상 문장으로 기록했다.
- [x] 세 파일과 기록을 hello-sw 저장소에 push 했다.
- [x] LMS에서 과제 1 요강을 확인했다 (마감: 4주차 수업 시작 전).

오늘의 출석 인증: 세 파일이 올라간 저장소 주소를 LMS 3주차 게시판에 제출하세요.

3강 · 정리

오늘의 세 문장

1. 생성형 AI는 다음에 올 글 조각을 확률로 고릅니다. 그래서 같은 지시에도 답이 달라질 수 있고, 결과는 언제나 검증 대상입니다.

2. 내가 비운 칸은 AI가 그럴듯하게 채웁니다. 역할 · 맥락 · 요구 · 형식 네 기둥으로 중요한 빈칸을 내가 먼저 채웁니다.

3. 오늘 세 지시문의 결과 차이가 그 증거입니다. 과제 1에서 나의 기능으로 같은 실험을 재현하고, 차이를 근거와 함께 분석합니다.

예고 · 4강

다음 주: 지시문 심화, 딸깍의 경계

예시로 가르치기

말로 설명하기 어려운 형식·스타일은 좋은 예시 한두 개가 문장 열 개를 이깁니다.

큰 작업을 단계로 나누기 (맛보기)

한 번에 시키면 왜 실패하는지. 7주차 본격 학습 전 미리 감을 잡습니다.

검증을 지시문에 담기

확인 코드와 통과 조건을 지시문 단계에서 함께 요구하는 법.

잊지 마세요

과제 1 마감: 4주차 수업 시작 전.

그리고 5주차에 팀 프로젝트 팀 구성과 주제 선정이 있습니다. 만들고 싶은 작은 서비스 아이디어를 미리 구상해 오세요.

오늘 수업은 여기까지. 저장소 주소 제출 확인하고 가세요. 수고하셨습니다!