PYTHON × AI VIBE CODING · 5시간 실습 교재

파이썬으로 완성하는 AI 바이브 코딩

Claude Code 중심 · 읽기 → 수정 → 검증 → 업무 적용

AI가 만든 코드를 "그대로 쓰는 사람"에서
"의도대로 고치고 책임지는 사람"으로

저자 윤재웅 · www.kpredict.co.kr

5시간 과정 (설명 55% · 데모 20% · 실습/토론 25%)

강의용 슬라이드 45장을 읽기 쉬운 문서 형태로 다시 엮었습니다.

이 PDF는 공개용으로 간추린 요약본입니다. 실제 강의 교재는 PowerPoint로 화면 캡처와 단계별 설명이 담겨 훨씬 자세합니다.

오늘의 도착점

5시간 뒤, 학습자가 직접 할 수 있어야 하는 것

목표	내용
① 설명한다	바이브 코딩의 개념과 Claude Code의 작동 방식을 비개발자 관점에서 설명한다.
② 읽고 고친다	AI가 만든 Python 코드에서 변수·조건·반복·함수·라이브러리를 찾아 최소 수정한다.
③ 검증한다	오류 메시지, 테스트, diff를 이용해 결과가 요구사항과 맞는지 확인한다.
④ 업무에 적용한다	엑셀 취합, 데이터 분석, 보고서, 웹 앱, 자동화의 구현 흐름을 설계한다.
⑤ 안전하게 사용한다	회사 데이터, API 키, 권한, 오픈소스, 배포 전 검토 기준을 적용한다.

5시간 운영안

파트	주제	시간
PART 1	바이브 코딩 이해	45분
PART 2	Python 읽기·수정	85분
PART 3	Claude Code 실전	80분
PART 4	업무 사례·최신 AI	60분
PART 5	보안·종합 실습	30분

PART 1

바이브 코딩, 정확히 이해하기

자연어로 만들되, 결과 중심의 빠른 반복과 사람의 검증을 결합하는 개발 방식입니다.

바이브 코딩이란?

아이디어를 자연어로 설명하고 AI가 코드를 생성·수정하는 협업 방식

사용자의 역할	AI의 역할	함께 해야 하는 일
문제와 사용자를 정의	프로젝트 구조 제안	작은 단위로 반복
입력 데이터와 출력 형태를 설명	여러 파일의 코드 작성	변경점을 확인
우선순위와 제약조건을 결정	오류 원인 탐색과 수정	실행 결과를 검증
결과가 맞는지 판단	테스트·문서 초안 생성	위험 작업은 승인 후 수행

> 핵심: "프롬프트 한 번"이 아니라 목표 → 생성 → 실행 → 피드백 → 수정 루프

왜 Python이 바이브 코딩과 잘 맞을까?

읽기 쉬운 문법 + 풍부한 라이브러리 + 빠른 실행

사람이 읽기 쉬움

들여쓰기와 비교적 간결한 문법으로 AI 결과를 검토하기 쉽다.

업무 데이터에 강함

pandas·openpyxl로 표, CSV, Excel을 다루기 좋다.

자동화에 강함

파일, 폴더, 이메일, 웹, API를 연결하는 스크립트에 적합하다.

AI 생태계의 중심

LLM API, RAG, 에이전트, 데이터 과학 도구가 Python을 폭넓게 지원한다.

"말로만 수정"할 때의 한계

AI가 만든 결과를 직접 고칠 수 있어야 하는 이유

상황	AI에게 계속 말로만 지시	Python을 읽을 수 있을 때
버튼 색만 변경	전체 화면을 다시 만들 수 있음	color 한 줄만 수정
합계 → 평균	요청 해석이 흔들릴 수 있음	`sum()` 을 `mean()` 으로 수정
오류 발생	같은 질문을 반복	Traceback의 마지막 줄 확인
기능 추가	기존 기능 회귀 가능	함수 단위 변경 + 테스트

Claude Code란?

코드베이스를 읽고, 파일을 편집하며, 명령을 실행하는 에이전틱 코딩 도구

프로젝트 이해

한 파일의 문답을 넘어 여러 파일과 폴더의 관계를 탐색합니다.

직접 작업

필요한 파일을 만들고 수정하며, 명령을 실행해 결과를 확인합니다.

개발 도구 연결

터미널, IDE, 데스크톱, 웹 등 여러 환경에서 사용할 수 있습니다.

역할 분담

사용자는 "무엇을 만들지"와 "무엇을 허용할지"를 정하고, Claude Code는 계획·수정·검증을 수행합니다.

Claude 대화형 코딩 vs Claude Code

어디에서 무엇까지 실행하는가가 다르다

Claude 채팅에서 코드 요청	Claude Code에서 작업 요청
✓ 설명·예제·코드 블록 생성 ✓ 아이디어 탐색과 빠른 질의 △ 프로젝트 전체 맥락은 별도 제공 △ 파일 반영과 실행은 사용자가 수행	✓ 코드베이스 탐색 ✓ 여러 파일 생성·수정 ✓ 터미널 명령과 테스트 실행 ✓ 변경점(diff) 검토 흐름
적합: 학습, 코드 설명, 작은 함수 초안	적합: 실제 프로젝트, 자동화, 반복 수정

좋은 바이브 코딩 요청의 6요소

모호한 "예쁘게" 대신 검증 가능한 작업 명세

목표	사용자	입력	출력	규칙	완료 기준
무엇을 해결?	누가 사용?	어떤 데이터?	어떤 화면/파일?	계산·예외?	어떻게 검증?

> **예** "sales.xlsx를 읽고, 지역별 매출 합계와 월별 추이를 보여주는 Streamlit 앱을 만들고, 빈 값은 제외하며, 샘플 테스트까지 작성해줘."

PART 2

AI 코드를 읽고 고치는 최소 문법

문법을 외우는 대신 "어디를 바꾸면 결과가 달라지는가"를 찾는 훈련을 합니다.

코드는 위에서 아래로 읽는다

주석 → 입력 → 처리 → 출력

```
# 1. 입력
price = 12000
qty = 5

# 2. 처리
total = price * qty

# 3. 출력
print(f'합계: {total:,}원')
```

- **수정 포인트 1** price 또는 qty 값을 바꾸면 결과가 달라집니다.
- **수정 포인트 2** `*`를 `+`로 바꾸면 계산 의미가 완전히 달라집니다.
- **수정 포인트 3** print 안의 글자와 표시 형식을 바꿀 수 있습니다.

변수와 자료형

AI 코드에서 가장 먼저 찾아야 할 "바꿀 수 있는 값"

```
company = 'KPREDICT'   # str
year = 2026            # int
growth = 0.125         # float
is_valid = True        # bool

print(company)
print(year, growth, is_valid)
```

문자열 str	따옴표 안의 글자. 파일명, 열 이름, 메시지에 자주 사용
숫자 int / float	횟수·연도·금액·비율. 따옴표를 붙이면 계산이 달라짐
참/거짓 bool	옵션 켜기/끄기, 조건 판단에 사용

리스트와 딕셔너리

여러 값을 담고, 구조를 설명하는 두 가지 핵심 상자

```
regions = ['Korea', 'Japan', 'Indonesia']

sale = {
    'region': 'Indonesia',
    'amount': 3200000,
    'currency': 'KRW'
}

print(regions[0])
print(sale['amount'])
```

- **리스트 [ ]** 순서가 있는 여러 값. 파일 목록, 열 목록, 처리할 항목에 사용
- **딕셔너리 { }** 이름표(key)와 값(value)의 묶음. 설정, JSON, API 응답에 사용
- **직접 수정 팁** 열 이름이 틀렸다면 딕셔너리 key 또는 리스트 항목을 먼저 확인

조건문: 업무 규칙을 코드로 읽기

if / elif / else는 "어떤 경우에 무엇을 할지"를 표현

```
amount = 12500000

if amount >= 10000000:
    level = 'High'
elif amount >= 5000000:
    level = 'Medium'
else:
    level = 'Low'

print(level)
```

- **비교 연산자** >=, <=, ==, != 가 기준을 어떻게 나누는가?
- **경계값** 정확히 10,000,000일 때 어느 구간인가?
- **들여쓰기** 조건에 포함되는 코드 블록의 범위를 확인

반복문: 같은 작업을 여러 번

파일·행·시트·이메일을 하나씩 처리하는 패턴

```
files = ['jan.csv', 'feb.csv', 'mar.csv']

for file in files:
    print(f'처리 시작: {file}')
    # 여기에 읽기/변환/저장 코드가 들어감

print('모든 파일 처리 완료')
```

> 실무 질문: "중간 파일 하나가 실패하면 전체를 멈출까, 건너뛸까?"

함수: 수정 범위를 작게 만드는 장치

입력 → 처리 → 반환을 하나의 이름으로 묶는다

```
def calculate_amount(price, qty, discount=0):
    subtotal = price * qty
    return subtotal * (1 - discount)

result = calculate_amount(12000, 5, 0.1)
print(result)
```

- **함수 이름** 무슨 일을 하는지 설명합니다.
- **매개변수** AI에게 바꿀 수 있는 입력을 분리하게 요청합니다.
- **return** 함수 밖에서 다시 쓸 결과를 돌려줍니다.

라이브러리와 import

직접 만들지 않고 검증된 도구를 불러오는 방식

```
from pathlib import Path
import pandas as pd

folder = Path('data')
files = list(folder.glob('*.xlsx'))

df = pd.read_excel(files[0])
print(df.head())
```

- **pathlib** 파일·폴더 경로를 운영체제에 맞게 다룸
- **pandas as pd** 표 데이터를 읽고, 필터·요약·집계
- **주의** AI가 제안한 패키지의 출처·버전·라이선스를 확인

에러 메시지는 답안지다

Traceback은 아래에서 위로 읽는다

```
Traceback (most recent call last):
  File "sales.py", line 8, in <module>
    total = df['매출액'].sum()
  File "pandas/core/frame.py", line ...
KeyError: '매출액'
```

1. **마지막 줄** 오류 종류와 핵심 원인: KeyError
2. **내 파일 위치** sales.py 8번째 줄을 먼저 확인
3. **실제 데이터** 열 이름·공백·오탈자 확인

AI에게 오류를 잘 전달하는 법

"안 돼요" 대신 재현 가능한 정보 제공

> 목표: sales.xlsx의 지역별 매출 합계를 계산
> 실행 명령: python sales.py
> 발생 위치: sales.py 8번째 줄
> 전체 오류: KeyError: '매출액'
> 확인한 것: 실제 열 이름은 '매출액 '처럼 뒤에 공백이 있음
> 요청: 원인을 설명하고 최소 변경으로 수정한 뒤 테스트 방법 제시

> 원칙: 목표 + 실행 방법 + 전체 오류 + 이미 확인한 것 + 원하는 수정 범위

가상환경과 패키지 버전

"내 PC에서는 되는데…"를 줄이는 기본 습관

```
# 프로젝트 폴더에서
python -m venv .venv

# Windows
.venv\Scripts\activate

# 패키지 설치
pip install pandas openpyxl streamlit

# 버전 기록
pip freeze > requirements.txt
```

- **격리** 프로젝트마다 필요한 패키지 버전을 분리
- **재현** requirements.txt로 동료 환경을 다시 구성
- **보안** 승인되지 않은 패키지 설치 여부 확인

미니 실습 · AI 코드 직접 수정

3분 안에 결과 바꾸기

```
import pandas as pd

df = pd.DataFrame({
    '지역': ['한국', '일본', '인도네시아'],
    '매출': [120, 90, 150]
})

result = df['매출'].sum()
print(f'전체 매출: {result}')
```

- **과제 A** 합계를 평균으로 바꾸기
- **과제 B** 120 이상인 지역만 필터링
- **과제 C** 출력 문구에 단위를 추가

PART 3

Claude Code로 계획하고, 만들고, 검증하기

좋은 결과는 강한 모델보다 좋은 작업 구조와 피드백 루프에서 나옵니다.

Claude Code 기본 작업 흐름

탐색 → 계획 → 구현 → 실행 → 검토

1 탐색	2 계획	3 구현	4 실행	5 검토
파일·구조·규칙 읽기	변경 범위와 위험 설명	작은 단위로 코드 수정	테스트·명령으로 확인	diff와 결과 승인

> **권장 지시** "먼저 파일을 읽고 계획만 제안해줘. 내가 승인하면 구현해."

프로젝트 컨텍스트 만들기

AI가 매번 같은 규칙을 잊지 않게 하기

README.md	CLAUDE.md	requirements.txt / pyproject
프로젝트 목적	코딩 스타일	패키지 버전
실행 방법	금지 작업	실행 환경
입력/출력 예시	테스트 명령	재현 가능한 설치
주요 폴더 구조	파일별 책임	의존성 변경 이력
알려진 제약	리뷰 체크리스트	

> 컨텍스트 엔지니어링 = AI에게 "무엇을 기억하고 어떤 기준으로 일할지"를 설계하는 일

한 번에 크게 만들지 않는다

작업을 수직 슬라이스로 나누기

데이터 읽기	sales.xlsx 10행으로 먼저 성공
핵심 계산	지역별 합계 함수와 테스트
화면 표시	표와 차트 1개
예외 처리	빈 파일·잘못된 열 이름
배포 준비	설정 분리·문서·권한 검토

PRD를 먼저 쓰면 결과가 달라진다

요구사항 정의서 예시

```
# 판매 분석 앱 PRD
사용자: 영업 담당자
문제: 월별 파일을 수작업으로 합치고 있음
입력: data/*.xlsx, 필수 열=일자/지역/매출
기능: 병합, 결측 제거, 지역별 집계, 월별 추이
출력: merged.csv + Streamlit 대시보드
예외: 빈 파일, 열 누락, 날짜 오류를 사용자에게 표시
완료 기준: 샘플 데이터 테스트 통과, 실행 명령 문서화
```

먼저 질문을 남기고 PRD를 완성한 뒤 구현하도록 요청하면, 중간 재작업을 줄일 수 있습니다.

diff를 읽는 최소 기술

AI가 바꾼 것만 확인하는 가장 빠른 방법

```
- total = df['매출'].sum()
+ total = df['매출'].mean()

- df = pd.read_excel('sales.xlsx')
+ df = pd.read_excel(input_file)
+ required = {'지역', '매출'}
+ if not required.issubset(df.columns):
+     raise ValueError('필수 열이 없습니다')
```

- **- 삭제된 줄** 기존 동작에서 무엇이 사라졌는가?
- **+ 추가된 줄** 새 동작과 오류 가능성은 무엇인가?
- **변경 범위** 요청하지 않은 파일까지 바뀌지 않았는가?

테스트가 바이브 코딩의 안전망

"작동해요"를 자동으로 확인 가능한 문장으로

```
def calculate_total(values):
    return sum(values)

def test_calculate_total():
    assert calculate_total([100, 200]) == 300
    assert calculate_total([]) == 0

# 실행: pytest -q
```

- **정상 케이스** 기대하는 입력에서 기대값이 나오는가?
- **경계 케이스** 빈 값, 0, 기준값과 같은 경우는?
- **오류 케이스** 잘못된 열·파일·형식에 친절히 실패하는가?

Claude Code 프롬프트 3종

상황에 따라 지시문을 바꾼다

> **A · 탐색** "이 프로젝트의 목적, 실행 흐름, 핵심 파일 5개, 위험한 외부 연동을 정리해줘. 파일은 수정하지 마."

> **B · 구현** "PRD를 기준으로 1단계 데이터 병합만 구현해. 기존 함수는 유지하고, 테스트와 실행 명령을 함께 제시해."

> **C · 리뷰** "방금 변경을 코드 리뷰해. 보안, 데이터 손실, 예외 처리, 회귀 가능성을 우선순위로 정리하고 직접 수정하지 마."

실습 시나리오 · Excel 12개 파일 병합

Claude Code + Python + pandas

1 요구 정의	필수 열·중복·날짜·출력 결정
2 샘플 확인	파일 2개와 열 이름 확인
3 PRD 작성	질문을 남기고 사용자 답변 반영
4 병합 구현	함수화 + 로그 + 오류 처리
5 검증	행 수·합계·중복·누락 비교
6 공유	CSV/Excel + 사용 안내

잘 안될 때 복구하는 5단계

대화가 꼬이면 컨텍스트를 줄이고 기준점으로 돌아간다

1 중지	추가 수정을 멈춘다
2 상태 확인	git diff / 변경 파일 확인
3 재현	오류 명령과 입력을 고정
4 최소 수정	원인 한 가지씩 변경
5 기준점 저장	작동하면 commit/tag

PART 4

업무 활용 사례와 최신 AI 기술

Python 코드를 읽는 능력은 데이터·앱·에이전트·RAG를 연결하는 공통 기반입니다.

활용 지도 · Python × Vibe Coding

작은 자동화에서 에이전트까지

데이터	문서	웹 앱	업무 자동화	AI 활용	에이전트
Excel/CSV 병합 정제·집계·차트	보고서 초안 파일명 정리· 변환	Streamlit 대시보드 간단한 입력 앱	폴더 감시 정기 실행· 알림	LLM API 요약·분류· 추출	도구 호출 RAG·MCP· 워크플로

사례 1 · 판매 데이터 분석 자동화

월별 파일 → 정제 → 지표 → 대시보드

월별 Excel	pandas	업무 지표	Streamlit
파일 규칙 검사	병합·결측·중복	지역/품목/월 집계	필터·차트·다운로드

> 직접 수정 포인트: 열 이름 · 필터 기준 · 집계 함수 · 차트 종류 · 출력 파일명

사례 2 · 문서와 보고서 자동 생성

구조화된 데이터에서 반복 보고를 줄인다

입력	처리	검증
Excel 실적 회의 메모 템플릿 파일 회사 용어집	데이터 요약 표·차트 생성 LLM으로 문장 초안 템플릿에 삽입	숫자 원본 대조 출처 표시 민감정보 제거 최종 승인

사례 3 · 사내 지식 검색 RAG

질문 → 검색 → 근거 → 답변

질문	검색	보강	생성	검증
사용자 의도	관련 문서 조각	프롬프트에 근거 추가	근거 기반 답변	출처·권한 확인

Python의 역할: 문서 전처리 · 메타데이터 · 임베딩 호출 · 검색 결과 가공 · 평가 자동화

최신 기술 1 · Agentic Coding

답변 생성에서 목표 달성 루프로

목표	계획	도구	관찰	수정
무엇을 끝낼까	작업을 나누기	파일·명령·검색	결과와 오류	다음 행동 결정

최신 기술 2 · MCP

AI 애플리케이션과 외부 데이터·도구를 연결하는 개방형 표준

MCP Host	Claude Code, IDE 등 AI 애플리케이션이 연결을 관리
MCP Client	각 서버와 전용 연결을 유지하는 구성요소
MCP Server	파일, 데이터베이스, 검색, 업무 도구를 제공

> 중요: 연결 가능성은 곧 권한·승인·감사의 필요성을 의미합니다.

최신 기술 3 · 멀티에이전트와 Skills

한 에이전트가 모든 일을 하지 않게 역할을 나눈다

Planner	요구사항을 작업으로 분해
Builder	코드와 파일을 구현
Reviewer / Tester	독립적으로 검토하고 테스트

Skills: 반복 업무의 지식·절차·템플릿을 재사용 가능한 형태로 제공

PART 5

기업 환경에서 안전하게 바이브 코딩하기

빠르게 만드는 것보다, 데이터·권한·변경·배포를 통제하는 것이 우선입니다.

바이브 코딩 보안 체크리스트

입력 전 · 실행 전 · 배포 전

입력 전	실행 전	배포 전
☐ 기밀·개인정보 제외	☐ 파일 삭제/덮어쓰기 확인	☐ 코드 리뷰와 테스트
☐ 가명/샘플 데이터 사용	☐ 외부 전송·네트워크 확인	☐ 라이선스 확인
☐ API 키를 코드에 직접 입력하지 않기	☐ 패키지 출처와 권한 확인	☐ 로그·감사·오류 처리
☐ 승인된 저장 위치 사용	☐ 최소 권한으로 실행	☐ 운영 승인과 롤백 계획

AI가 만든 코드 리뷰 기준 8가지

정답이 아니라 "검토 가능한 초안"으로 다룬다

1 요구사항 충족	2 데이터 손실 위험	3 권한·비밀정보	4 오류·예외 처리
5 테스트 존재	6 의존성·라이선스	7 성능·확장성	8 로그·운영 가능성

> "코드가 실행된다"와 "업무에 안전하게 쓸 수 있다"는 다른 기준입니다.

종합 실습 · 나의 업무를 1개 설계하기

구현보다 "좋은 시작 문서" 완성이 목표

1 문제	지금 반복되는 수작업은?
2 입력	파일·열·데이터 출처는?
3 규칙	계산·필터·예외는?
4 출력	파일·화면·알림 형태는?
5 완료 기준	정확성을 어떻게 확인?
6 안전 기준	넣으면 안 되는 데이터는?

마무리 · 바이브 코딩의 경쟁력

생성 속도가 아니라 정의·수정·검증·책임

DEFINE	READ	EDIT	VERIFY	GOVERN
문제를 구조화한다	AI 코드를 읽는다	최소 범위를 수정한다	테스트와 근거로 확인한다	안전하게 운영한다

Python을 알면 AI의 결과를 직접 고칠 수 있고, 검증할 수 있으며, 더 정확하게 지시할 수 있습니다.

> 다음 행동: 내 업무 1개를 샘플 데이터로 작게 시작하고, 작동 기준점을 저장하세요.