AI활용프로그래밍

Week 7. 튜플·딕셔너리·집합

자료구조 확장 + With AI 문제 해결 연습

학습 목표

- 튜플(tuple), 딕셔너리(dict), 집합(set)의 특징을 설명할 수 있다.
- 각 자료구조의 기본 문법/메서드를 활용할 수 있다.
- 문제에 맞는 자료구조를 선택하는 기준을 이해한다.
- LLM을 활용해 자료구조 기반 프로그램을 설계·구현·개선할 수 있다.

오늘의 구성

1) 튜플(tuple) 기초
2) 딕셔너리(dict) 기초
3) 집합(set) 기초
4) 자료구조 선택 가이드
5) With AI 실습 + 퀴즈/과제

튜플(tuple)란?

- 여러 값을 순서대로 저장(리스트와 유사)
- 하지만 수정 불가(immutable)
- 의미: “바뀌면 안 되는 데이터”에 적합
- 예: 좌표(x,y), RGB 색상, (이름, 학번)

튜플 기본 예시

```
point = (10, 20)
x, y = point
print(x, y)

# swap (값 교환)
a, b = 1, 2
a, b = b, a
print(a, b)
```

핵심 포인트

- 패킹/언패킹(pack/unpack)
- 튜플은 함수에서 여러 값을 반환할 때 자주 사용
- 값 교환(swap)이 매우 간단

튜플 vs 리스트

- 리스트: 수정/추가/삭제가 필요할 때
- 튜플: 고정된 값 묶음(상수처럼)으로 사용할 때
- 튜플이 약간 더 가볍고 안전(실수로 변경 방지)
- "변경 가능해야 하는가?"가 선택 기준

딕셔너리(dict)란?

- key-value(키-값) 구조의 자료구조
- 키로 값을 빠르게 찾는 “검색”에 강함
- 예: {"Kim":90, "Lee":80}
- 현실 예: 학번→이름, 단어→빈도수

dict 생성/접근/수정

```
score = {"Kim": 90, "Lee": 80}
print(score["Kim"])  # 90

score["Park"] = 70   # 추가
score["Lee"] = 85    # 수정
print(score)
```

핵심 포인트

- 키가 없는데 score["X"]를 하면 KeyError
- get()을 쓰면 기본값 지정 가능
- 딕셔너리는 순서보다 "매핑"이 핵심

dict 메서드(get/items)

```
score = {"Kim": 90}
print(score.get("Lee", 0))

for name, s in score.items():
    print(name, s)
```

핵심 포인트

- get(key, default): 키가 없을 때 기본값
- items(): (key,value) 쌍 순회
- keys()/values()도 자주 사용

dict 자주 쓰는 기능 3가지

- 키 존재 확인: if key in d:
- 삭제: pop(key) 또는 del d[key]
- 여러 값 추가/병합: update(other_dict)
- 실무 팁: “키가 없을 가능성”부터 먼저 고려

빈도수 세기(대표 예제)

```
text = "banana"
freq = {}

for ch in text:
    freq[ch] = freq.get(ch, 0) + 1

print(freq)  # {'b':1,'a':3,'n':2}
```

핵심 포인트

- 딕셔너리의 "가장 유명한" 사용법
- get으로 기본값 0 처리
- 단어 빈도/로그 분석/통계에 활용

집합(set)란?

- 중복을 허용하지 않는 자료구조
- 순서가 중요하지 않음
- 주요 기능: 중복 제거, 포함 여부 검사(in)
- 수학의 집합 연산(합집합/교집합/차집합)

set 중복 제거

```
nums = [1, 1, 2, 3, 3]
s = set(nums)
print(s)  # {1,2,3}

nums2 = list(s)
print(nums2)
```

핵심 포인트

- set은 출력 순서가 매번 다를 수 있음
- 중복 제거 후 다시 리스트로 변환 가능
- 정렬이 필요하면 sorted(list(set(...)))

집합 연산(합/교/차)

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)  # union
print(a & b)  # intersection
print(a - b)  # difference
```

핵심 포인트

- |, &, - 연산자로 집합 연산
- 중복 없는 비교/필터에 강함
- 데이터 전처리(Week12)에서도 자주 등장

자료구조 선택 가이드

- list: 순서 O, 중복 O, 수정 O
- tuple: 순서 O, 중복 O, 수정 X
- dict: key로 빠른 조회(매핑)
- set: 중복 X, 포함 검사/집합 연산

With AI: “자료구조 선택” 질문법

- 문제 설명 + 입력 데이터 예시 + 출력 형태를 제공
- “어떤 자료구조가 적절한지” 먼저 물어보기
- 그 다음에 구현을 요청(코드 생성)
- 마지막에 리팩터링/엣지 케이스 보완 요청

프롬프트 템플릿(자료구조 선택)

```
역할: 파이썬 설계 코치
문제: 학생 이름과 점수를 저장하고, 이름으로 점수를 조회
입력 예시: Kim 90, Lee 80
기능: add, find, list
질문: list/tuple/dict/set 중 무엇이 적절? 이유는?
그 후: 선택한 구조로 코드 작성 + 테스트
```

프롬프트 포인트

- AI에게 "선택 이유"를 말하게 하면 학습 효과↑
- 선택→구현 순서로 요청
- 테스트 케이스를 꼭 요구

실습 1: 연락처 관리(dict)

- 기능: add(이름/번호), find(이름), list(전체 출력)
- dict 사용: name → phone
- 도전: 같은 이름이 있으면 "수정"할지 "거절"할지 정책 정하기
- LLM 활용: 정책을 명시하고 코드 생성 요청

실습 2: 단어 빈도 분석(dict)

- 문장 입력 → 단어 리스트로 분해(split)
- dict로 단어 빈도수 세기
- 출력: 가장 많이 나온 단어 TOP 3 (도전)
- LLM 활용: “정렬/상위 N” 구현 아이디어 얻기

실습 3: 로또 번호 검사(set)

- 사용자 입력 6개(중복 없이) → set으로 저장
- 당첨 번호 set과 교집합으로 맞춘 개수 계산
- 주의: 중복 입력 처리(중복이면 다시 입력)
- LLM 활용: 입력 검증 로직(while) 설계 도움받기

AI 답변 검증 체크리스트

- dict에서 KeyError가 나지 않게 get/검증을 했나?
- set은 순서가 없다는 점을 고려했나?
- 요구한 정책(중복 처리/수정 여부)이 코드에 반영됐나?
- 테스트 5개 이상 실행했나?
- 자료구조 선택 이유를 설명할 수 있나?

미니 퀴즈(5문항)

1) tuple의 가장 큰 특징은?
2) dict에서 키가 없을 때 안전하게 값을 얻는 방법은?
3) set이 중복 제거에 좋은 이유는?
4) (a,b) = (b,a)가 가능한 이유는?
5) list vs dict 선택 기준 1가지는?

과제/연습문제

- 기본: 연락처 관리 프로그램(dict) 제출
- 도전: 단어 빈도 TOP-N 출력 + 동점 처리 규칙 만들기
- 제출: .py 1개 + 실행 캡처 1장
- AI 사용 시: 프롬프트 2개 + 선택한 자료구조 이유 기록

요약 & 다음 주 예고

- 오늘: tuple/dict/set 기초와 사용 장면
- 핵심: 자료구조는 "문제 성격"에 따라 선택
- 다음 주(Week 8): 중간 시험
- 시험 범위(예): Week 1~7 핵심 문법 + 실습 코드 읽기/수정