

Python

CH05: 파이썬을 활용한 프로젝트 만들기

전종준 교수

서울시립대학교 통계학과/통계데이터사이언스대학원

Contents

- 패키지의 설치 및 활용
- 중급 데이터 객체: 리스트 컴프리헨션과 언패킹
- 프로젝트: 가계부 프로그램 만들기

패키지의 설치 및 활용

패키지란?

- ▶ 다른 사람이 만들어 둔 코드 묶음 (예: 수학, 크롤링, 시각화 등)
- ▶ Python에서는 pip으로 설치 및 관리
- ▶ 설치 후 import로 사용 가능

Colab에서 패키지 설치하기

- ▶ 터미널 명령어 대신 코드 셀에서 실행

- ▶ 두 가지 방법:

- ▶ `!pip install 패키지이름`

(셸 명령어 방식)

- ▶ `%pip install 패키지이름`

(Jupyter 매직 명령어)

```
1 !pip install requests
```

설치한 패키지 사용하기

- ▶ import 패키지이름으로 불러오기
- ▶ 예시: requests로 웹 페이지 가져오기

```
1 import requests
2
3 response = requests.get("https://example.com")
4 print(response.status_code)
```

버전 확인과 업그레이드

- ▶ 버전 확인: `!pip show 패키지이름`
- ▶ 업그레이드: `!pip install --upgrade 패키지이름`
- ▶ Colab에서는 `%pip`로도 동일하게 가능

```
1 %pip show requests
2 %pip install --upgrade requests
```

실습: 날씨 정보 가져오기

- ▶ requests를 이용해 wttr.in에서 텍스트 기반 날씨 확인
- ▶ 결과는 간단한 문자열로 출력됨

```
1 !pip install requests
2
3 import requests
4
5 url = "https://wttr.in/seoul?format=3"
6 response = requests.get(url)
7 print(response.text)
```

import 구문

- ▶ 전체 모듈을 불러올 때 사용
- ▶ 모듈 내 모든 기능에 접근 가능
- ▶ 사용 시: 모듈이름.기능이름

```
1 import math
2
3 print(math.sqrt(16))
```

import as 구문

- ▶ 모듈에 별칭(alias)을 붙여 간단하게 사용
- ▶ 이름이 긴 모듈이나 자주 사용할 때 유용

```
1 import numpy as np
2
3 a = np.array([1, 2, 3])
4 print(a)
```

from ... import ... 구문

- ▶ 모듈에서 특정 함수나 클래스만 불러오기
- ▶ 모듈 이름 없이 함수 이름만으로 사용 가능
- ▶ 여러 개 불러올 때는 쉼표로 구분

```
1 from math import sqrt, pi
2
3 print(sqrt(25))
4 print(pi)
```

from ... import ... as ... 구문

- ▶ 특정 기능에 별칭 붙이기
- ▶ 이름이 길거나 겹칠 때 유용

```
1 from math import factorial as fact
2
3 print(fact(5))
```

중급 데이터 객체: 리스트 컴프리헨션과 언패킹

리스트 컴프리헨션과 언패킹 소개

▶ 리스트 컴프리헨션 (List Comprehension)

- ▶ 반복문을 한 줄로 간결하게 표현하는 방법.
- ▶ 기존 리스트를 변형하거나 새로운 리스트를 만들 때 사용.
- ▶ 예: `[x ^ 2 for x in range(5)]`

▶ 리스트 언패킹 (List Unpacking)

- ▶ 리스트의 여러 값을 각각 변수에 **한 번에** 담을 수 있음.
- ▶ * 기호로 일부만 받고 나머지를 묶을 수도 있음.
- ▶ 예: `a, *b = [1, 2, 3, 4]`

두 기술 모두 코드를 더 짧고, 읽기 쉽게 만들어준다.

리스트 컴프리헨션이란?

- ▶ 리스트를 간결하게 생성하는 방법이에요.
- ▶ for문을 한 줄로 줄일 수 있어요.
- ▶ 읽기 쉽고 빠르게 리스트를 만들 수 있어요.

형식:

1 [표현식 `for` 항목 `in` 반복가능한객체]

기본 예시: 제곱 리스트 만들기

- ▶ 1부터 5까지의 제곱 값을 리스트로 만들자.

예시 코드:

```
1 squares = [x**2 for x in [1, 2, 3, 4, 5]]
2 print(squares)
3 # 출력: [1, 4, 9, 16, 25]
```

- ▶ list Comprehension 사용하지 않고 for 문을 사용하여 1부터 5까지의 제곱 값을 리스트로 만들자.

예시 코드:

```
1 squares = []
2 for i in range(1,6):
3     squares.append(i**2)
```

조건문이 있는 리스트 컴프리헨션

- ▶ 짝수만 필터링해서 리스트로 만들 수 있어요.

예시 코드:

```
1 evens = [x for x in range(10) if x % 2 == 0]
2 print(evens)
3 # 출력: [0, 2, 4, 6, 8]
```

예시 코드:

```
1 evens = []
2 for x in range(10):
3     if x % 2 == 0:
4         evens.append(x)
5 print(evens)
6 # 출력: [0, 2, 4, 6, 8]
```

lambda 함수란?

- ▶ 간단한 함수를 한 줄로 만들 수 있는 방법이에요.
- ▶ 이름 없이 사용하는 익명 함수라고도 불러요.

형식:

```
1 lambda 매개변수: 표현식
```

예:

```
1 add = lambda x, y: x + y
2 print(add(3, 5)) # 출력: 8
```

lambda 함수 + 리스트 컴프리헨션

- ▶ 리스트 안에서 값을 계산할 때 lambda를 유용하게 쓸 수 있어요.
- ▶ 특히 map(), sorted(), filter() 같은 함수와 자주 함께 사용돼요.

예: 문자열 길이로 정렬하기

```
1 words = ["apple", "kiwi", "banana"]
2 sorted_words = sorted(words, key=lambda x: len(x))
3 print(sorted_words)
4 # 출력: ['kiwi', 'apple', 'banana']
```

예제: 길이가 5 이상인 단어를 대문자로

- ▶ 단어 리스트에서 길이가 5 이상인 단어만 골라 대문자로 바꿔볼게요.
- ▶ lambda로 변환 함수를 만들고, 조건은 if로 처리해요.

예시 코드:

```
1 words = ["cat", "tiger", "lion", "elephant"]
2 upper = lambda w: w.upper()
3 result = [upper(w) for w in words if len(w) >= 5]
4 print(result)
5 # 출력: ['TIGER', 'ELEPHANT']
```

예제: 세금 포함 가격 리스트 만들기

- ▶ 물건 가격 리스트가 있을 때, 각 항목에 10% 세금을 더한 새 리스트를 만들어요.
- ▶ lambda로 세금 계산 함수를 만들어요.

예시 코드:

```
1 prices = [1000, 2500, 3300]
2 with_tax = lambda p: int(p * 1.1)
3 final_prices = [with_tax(p) for p in prices]
4 print(final_prices)
5 # 출력: [1100, 2750, 3630]
```

리스트 언패킹이란?

- ▶ 리스트의 요소를 각각 변수에 **한 번에 할당**할 수 있어요.
- ▶ `a, b = [1, 2]` 처럼 사용할 수 있어요.

예시:

```
1 data = [10, 20]
2 x, y = data
3 print(x)  # 출력: 10
4 print(y)  # 출력: 20
```

* 연산자 응용: 여러 변수 받기

- ▶ 앞이나 중간, 끝에서 여러 값을 받을 수도 있어요.
- ▶ * 연산자는 하나만 쓸 수 있어요.

예시:

```
1 data = [10, 20, 30, 40, 50]
2 a, *b, c = data
3 print(a)    # 출력: 10
4 print(b)    # 출력: [20, 30, 40]
5 print(c)    # 출력: 50
```

enumerate()와 함께 언패킹

- ▶ enumerate()는 (인덱스, 값) 튜플을 반환해요.
- ▶ for문에서 바로 두 값을 언패킹해서 쓸 수 있어요.

예시: 목록 번호와 함께 출력

```
1 items = ["egg", "milk", "bread"]
2
3 for idx, item in enumerate(items):
4     print(f"{idx + 1}. {item}")
```

딕셔너리 순회 시 언패킹

- ▶ `dict.items()`는 (키, 값) 쌍을 반환.
- ▶ 키와 값을 각각 변수에 할당할 수 있어요.

예시: 학생 점수 출력

```
1 scores = {"Alice": 85, "Bob": 90}
2
3 for name, score in scores.items():
4     print(f"{name}의 점수는 {score}점")
```

프로젝트: 가계부 프로그램 만들기

프로젝트 소개: 가계부 프로그램

- ▶ 목표: 수입과 지출을 기록하고 관리하는 프로그램 만들기
- ▶ 기능:
 - ▶ 항목 추가 (날짜, 내용, 금액, 분류)
 - ▶ 기록 보기
 - ▶ 통계 보기 (총합, 분류별 합계 등)
 - ▶ 파일 저장 및 불러오기

데이터 구조 설계

- ▶ 각 거래 기록을 dict로 표현
- ▶ 여러 개의 기록을 list에 저장

예시:

```
1 record = {  
2     "date": "2025-06-29",  
3     "category": "식비",  
4     "description": "편의점",  
5     "amount": -3500  
6 }  
7  
8 ledger = [record]
```

사용자 입력 받아 기록 추가하기

- ▶ `input()`을 사용해 항목 정보 입력받기
- ▶ 문자열을 적절히 가공하여 `dict`로 저장

```
1 date = input("날짜를 입력하세요 (예: 2025-06-29): ")
2 category = input("분류를 입력하세요 (예: 식비): ")
3 desc = input("내용을 입력하세요: ")
4 amount = int(input("금액을 입력하세요 (+수입, -지출): "))
5
6 ledger.append({
7     "date": date,
8     "category": category,
9     "description": desc,
10    "amount": amount
11 })
```

기록 출력하기

- ▶ 리스트에 저장된 모든 기록 출력
- ▶ 반복문과 문자열 포매팅 활용

```
1 for record in ledger:
2     print(f"{record['date']} | {record['category']} | "
3           f"{record['description']} | {record['amount']}원")
```

총합 계산하기

- ▶ 전체 금액의 합계를 계산
- ▶ 리스트 컴프리헨션을 사용하면 코드가 간결해짐

```
1 total = sum([r["amount"] for r in ledger])  
2 print(f"전체 합계: {total}원")
```

분류(category)별 합계 구하기

- ▶ dict를 이용해 카테고리별로 누적합 저장

```
1 summary = {}
2 for r in ledger:
3     cat = r["category"]
4     summary[cat] = summary.get(cat, 0) + r["amount"]
5
6 for cat, amount in summary.items():
7     print(f"{cat}: {amount}원")
```

pickle로 파일 저장하기

- ▶ pickle은 파이썬 객체를 그대로 저장하는 바이너리 방식
- ▶ 리스트나 딕셔너리 형태를 그대로 보존
- ▶ 사람이 직접 보기엔 어렵지만, 프로그램엔 더 편리

```
1 import pickle
2
3 with open("ledger.pkl", "wb") as f:
4     pickle.dump(ledger, f)
```

입력 기능을 함수로 만들기

- ▶ 재사용 가능한 함수로 분리하면 코드가 깔끔해져요

```
1 def add_record():
2     try:
3         date = input("날짜: ")
4         category = input("분류: ")
5         desc = input("내용: ")
6         amount = int(input("금액: "))
7         return {"date": date, "category": category,
8                 "description": desc, "amount": amount}
9     except ValueError:
10        print("금액은 숫자로 입력해주세요!")
11        return None
```

메뉴 구성과 전체 흐름 만들기

- ▶ while True로 반복 실행
- ▶ 메뉴 선택 → 기능 실행

```
1 while True:
2     print("1. 추가 2. 보기 3. 통계 4. 저장 5. 불러오기 0. 종료")
3     choice = input("선택> ")
4
5     if choice == "1":
6         r = add_record()
7         if r: ledger.append(r)
8     elif choice == "2":
9         for r in ledger:
10            print(r)
11    elif choice == "0":
12        break
```

통계 기능 추가하기

- ▶ `sum()`, `dict`로 전체/분류별 합계 계산
- ▶ 함수로 분리해서 메뉴에서 호출

```
1 def show_stats():
2     total = sum([r["amount"] for r in ledger])
3     print(f"전체 합계: {total}원")
4
5     summary = {}
6     for r in ledger:
7         cat = r["category"]
8         summary[cat] = summary.get(cat, 0) + r["amount"]
9
10    for cat, amount in summary.items():
11        print(f"{cat}: {amount}원")
```

통계 함수 자세히 보기: show_stats()

목표: 가계부 데이터에서 전체 합계와 분류별 합계를 출력하는 함수
코드 전체:

```
1 def show_stats():
2     total = sum([r["amount"] for r in ledger])
3     print(f"전체 합계: {total}원")
4
5     summary = {}
6     for r in ledger:
7         cat = r["category"]
8         summary[cat] = summary.get(cat, 0) + r["amount"]
9
10    for cat, amount in summary.items():
11        print(f"{cat}: {amount}원")
```

show_stats() 설명 (1/2)

- ▶ ledger는 거래 기록이 저장된 리스트
- ▶ 각 항목은 dict 구조로 되어 있음
- ▶ 먼저 전체 금액 합계 계산

```
1 total = sum([r["amount"] for r in ledger])  
2 print(f"전체 합계: {total}원")
```

- ▶ 리스트 컴프리헨션을 이용해 모든 거래의 금액(amount)만 추출
- ▶ sum()을 이용해 전체 합을 구함
- ▶ 결과 출력 시 f-string 사용

show_stats() 설명 (2/2)

- ▶ 분류(category)별 합계를 딕셔너리로 누적 저장

```
1 summary = {}
2 for r in ledger:
3     cat = r["category"]
4     summary[cat] = summary.get(cat, 0) + r["amount"]
5
6 for cat, amount in summary.items():
7     print(f"{cat}: {amount}원")
```

- ▶ summary.get(cat, 0)는 초기값이 없을 경우 0을 사용
- ▶ 마지막 반복문에서 각 카테고리별 누적 합계를 출력

파일 저장 및 불러오기 함수 (pickle)

- ▶ pickle 모듈을 활용해 전체 ledger 저장/복원

```
1 import pickle
2
3 def save_file():
4     with open("ledger.pkl", "wb") as f:
5         pickle.dump(ledger, f)
6     print("저장 완료")
7
8 def load_file():
9     global ledger
10    with open("ledger.pkl", "rb") as f:
11        ledger = pickle.load(f)
12    print("불러오기 완료")
```

메뉴 전체 통합

```
1 while True:
2     print("1. 추가 2. 보기 3. 통계 4. 저장 5. 불러오기 0. 종료")
3     choice = input("선택> ")
4     if choice == "1":
5         r = add_record()
6         if r: ledger.append(r)
7     elif choice == "2":
8         for r in ledger:
9             print(r)
10    elif choice == "3":
11        show_stats()
12    elif choice == "4":
13        save_file()
14    elif choice == "5":
15        load_file()
16    elif choice == "0":
17        break
18    else:
19        print("올바른 숫자를 입력해주세요.")
```