

Python

CH01: 파이썬 시작하기 I

전종준 교수

서울시립대학교 통계학과/통계데이터사이언스대학원

Contents

- Introduction
- Data Type
- Overview of Python Data Collections
- List
- 초급 넘어서기: 인덱스를 이용한 슬라이싱

Introduction

파이썬소개

파이썬이란

- ▶ 쉽고 간결한 문법의 프로그래밍 언어
- ▶ 배우기 쉬워 초보자에게 적합
- ▶ 다양한 분야에 사용 가능:
 - ▶ 데이터 분석, 인공지능
 - ▶ 웹 개발, 자동화 스크립트
 - ▶ 교육용 코딩 등

파이썬의 장점

- ▶ 문법이 읽기 쉽고 직관적
- ▶ 방대한 라이브러리 생태계
- ▶ 다양한 플랫폼에서 사용 가능
- ▶ 커뮤니티와 자료가 매우 풍부

간단한 파이썬 코드 예시

```
1  name = "Alice"  
2  age = 20  
3  print("이름:", name)  
4  print("나이:", age)
```

Google Colab이란?

- ▶ 구글에서 제공하는 무료 온라인 파이썬 실행 환경
- ▶ 설치 없이 웹 브라우저에서 바로 사용 가능
- ▶ 실시간 공동 작업, 자동 저장, GPU 지원 등

Colab 시작하기

1. 구글 계정으로 로그인
2. <https://colab.research.google.com> 접속
3. 새 노트북 만들기 클릭
4. 코드 셀에 파이썬 코드 작성 후 Shift + Enter 로 실행

Colab 기본 화면 구성

- ▶ **코드 셀:** 파이썬 코드 작성
- ▶ **텍스트 셀:** 설명 및 마크다운 입력
- ▶ **파일 메뉴:** 노트북 저장 및 공유
- ▶ **런타임 메뉴:** 실행 환경 제어 (재시작, 런타임 유형 변경 등)

실습: 첫 번째 파이썬 코드 실행해보기

```
1 print("안녕하세요, 파이썬!")
```

- ▶ 위 코드를 Colab에 입력해보고 실행해보세요.
- ▶ 단축키: Shift + Enter

Colab 단축키 안내 포인트

▶ 실행:

- ▶ Shift + Enter → 셀 실행 및 다음 셀로 이동

▶ 주석처리:

- ▶ Ctrl + / → 주석 처리 또는 해제

▶ 편집:

- ▶ Ctrl + M, A → 위에 코드 셀 추가
- ▶ Ctrl + M, B → 아래에 코드 셀 추가
- ▶ Ctrl + M, D, D → 셀 삭제
- ▶ 셀 복사/붙여넣기 (Ctrl + M, C, Ctrl + M, V)
- ▶ 셀 이동 (Ctrl + M, K/J)

여러 줄 코드 작성하기

- ▶ Colab 셀에서 여러 줄 코드를 입력할 수 있음.
- ▶ 자동 들여쓰기와 실행 단축키(Shift + Enter)를 익혀보자.

예시 코드:

```
1 name = "철수"
2 age = 12
3
4 print("이름:", name)
5 print("나이:", age)
```

셀 나누기와 합치기

- ▶ 긴 코드를 셀로 나눠본다: Ctrl + M + -
- ▶ 셀을 합치고 싶을 땐 Shift로 셀 다중 선택 후 마우스 오른쪽 Merge cells

예시 코드:

```
1 # 셀 1
2 a = 5
3 b = 3
4
5 # 셀 2
6 print("합:", a + b)
```

셀 복사, 붙여넣기, 이동

- ▶ 셀 복사: Ctrl + M + C, 붙여넣기: Ctrl + M + V
- ▶ 셀 위로 이동: Ctrl + M + K, 아래로 이동: Ctrl + M + J

연습: 아래 셀을 복사하고 실행 순서를 바꾸기.

```
1 print("첫 번째 출력")
2 print("두 번째 출력")
```

코드 셀 끼워넣기

- ▶ 새로운 코드 셀 추가:
 - ▶ 아래에 삽입: Ctrl + M + B
 - ▶ 위에 삽입: Ctrl + M + A
- ▶ 셀을 나눠서 간결한 실습 코드를 만들기

연습: 아래 코드를 두 개의 셀로 나눠서 실행.

```
1 x = 10
2 y = 5
3 print("곱셈 결과:", x * y)
```

Markdown 셀로 설명 달기

- ▶ 코드에 설명을 붙이려면 Markdown 셀을 사용하면 된다
- ▶ 셀을 선택하고 Ctrl + M + M을 누르면 Markdown 셀로 변경된다

예시 Markdown 내용:

곱셈 실습

두 수를 곱한 결과를 출력한다

이어질 코드 셀 예시:

```
1 x = 7
2 y = 8
3 print(x * y)
```

오타 수정하고 다시 실행하기

- ▶ 오류 메시지를 읽고, 직접 수정해보기
- ▶ 셀 수정 후 다시 실행: Shift + Enter

예시 코드 (고쳐야 할 부분이 있음!):

```
1 print(Hello, world)
```

런타임 재시작과 변수 초기화

- ▶ 변수와 모듈은 런타임에 저장돼 있어요.
- ▶ 에러가 계속 날 때는 런타임 재시작이 해결책이 될 수 있어요!
- ▶ 런타임 → 런타임 다시 시작 메뉴를 사용하거나 Ctrl + M + .

예시 코드:

```
1 message = "초기 메시지"  
2 print(message)
```

Data Type

데이터 타입이란?

- ▶ 데이터 타입은 "값의 종류와 성질"을 나타냅니다.
- ▶ 데이터 타입 (data type)
 - ▶ 10 → 정수 (int)
 - ▶ 3.14 → 실수 (float)
 - ▶ "hello" → 문자열 (str)
 - ▶ True, False → 논리형 (bool)
- ▶ 타입을 알면:
 - ▶ 무엇이 가능한지 알 수 있고
 - ▶ 오류를 방지할 수 있습니다

질문: 왜 데이터 타입을 구분할까?

- ▶ 숫자 + 숫자 = 계산 가능
- ▶ 문자열 + 문자열 = 연결 가능
- ▶ 숫자 + 문자열 = 오류 발생
- ▶ 서로 다른 종류의 데이터를 혼합하면 문제가 발생

질문

“3 더하기 사과”가 가능한가요?

- ▶ 사람은 이해 못함 → 컴퓨터도 마찬가지!
- ▶ 데이터 간의 “의미 있는 연산”을 위해 타입이 필요

타입은 기능을 결정한다

코드

```
1 x = "3"  
2 y = 4  
3 print(x + y)  # TypeError
```

- ▶ "3"은 문자열, 4는 숫자
- ▶ 서로 다른 타입끼리 연산 불가
- ▶ 해결: `int(x) + y`

(질문) 문자열을 더해보세요!

타입은 기능을 결정한다

예시

```
1 a = "hello"
2 a.upper()    # 가능 (문자열을 대문자로 바꾸는 기능)
3
4 b = 10
5 b.upper()    # 오류
```

- ▶ 문자열에는 문자열 함수 사용 가능
- ▶ 숫자에는 해당 기능 없음

타입은 데이터의 저장효율을 높인다.

예시1

```
1 import sys
2 x = 1000000000      # 정수
3 y = "1000000000"    # 문자열
4 print(sys.getsizeof(x)) # 정수형 크기
5 print(sys.getsizeof(y)) # 문자열 크기
```

예시2

```
1 a = 42              # int
2 b = 42.0            # float
3 print("정수 a:", sys.getsizeof(a), "bytes")
4 print("실수 b:", sys.getsizeof(b), "bytes")
```

타입은 데이터의 저장효율을 높인다.

예시3

```
1  x = True
2  y = False
3  print(sys.getsizeof(x))
4  print(sys.getsizeof(y))
```

예시4

```
1  print("빈 문자열:", sys.getsizeof(""), "bytes")
2  print("숫자 0:", sys.getsizeof(0), "bytes")
```

올바른 데이터 타입의 사용

데이터의 type 을 어떻게 확인하나?

- ▶ 파이썬에서는 `type()` 함수를 사용하여 데이터의 타입을 확인할 수 있다.
- ▶ `type()`은 객체의 종류(클래스)를 반환한다.

데이터의 type 을 변경 (type casting)해야 한다면?

- ▶ 파이썬의 기본 타입 간에는 서로 변환이 가능하다.
- ▶ 주로 사용하는 타입: `int`, `float`, `str`, `bool`
- ▶ 형식과 의미가 맞아야 변환이 성공함.

올바른 데이터 타입의 사용

기본 타입 간 형변환 표

From	To	예시	결과
int	float	<code>float(3)</code>	3.0
float	int	<code>int(3.9)</code>	3
int/float	str	<code>str(10.5)</code>	"10.5"
str	int	<code>int("5")</code>	5
str	float	<code>float("3.14")</code>	3.14
int/float	bool	<code>bool(0)</code>	False
str	bool	<code>bool("")</code>	False
bool	int	<code>int(True)</code>	1
bool	str	<code>str(False)</code>	"False"

Table: 데이터 타입 변환 함수의 사용 예시

올바른 데이터 타입의 사용

형변환 예제 코드

```
1 print(int(3.7))           # 3
2 print(float("2.5"))       # 2.5
3 print(str(True))          # "True"
4 print(bool(0))             # False
5 print(bool(""))            # False
6 print(bool("hello"))       # True
7 print(int(True))           # 1
8 print(float(False))        # 0.0
```

문제 1. (객관식)

다음 중 `type("123")`의 출력 결과로 옳은 것은?

1. `<class 'int'>`
2. `<class 'float'>`
3. `<class 'bool'>`
4. `<class 'str'>`

문제 2. (서술형)

다음 질문에 답하세요.

정수형 값 123456789와 문자열 "123456789"를 `sys.getsizeof()`로 비교했을 때, 왜 문자열이 더 많은 메모리를 차지할까요?

문제 3. (객관식)

Google Colab에서 코드 셀을 실행하고 다음 셀로 이동하는 단축키는?

1. Ctrl + Enter
2. Shift + Enter
3. Alt + Enter
4. Ctrl + Shift + Enter

문제 4. (서술형)

아래 코드를 보고 실행 결과와 데이터 타입을 설명하시오.

```
x = float("3.14")  
print(x)  
print(type(x))
```

Overview of Python Data Collections

데이터 객체

- ▶ 데이터를 저장하고 다루는 방식의 '그릇'
- ▶ 상황에 따라 적절한 데이터 객체를 사용하는 것이 매우 중요함
- ▶ 대표적인 데이터 객체: 리스트(list), 딕셔너리(dictionary), 튜플(tuple), 셋(set)

리스트 (List)

실용 예제

TODO 리스트, 쇼핑 목록

개발 예시

순서대로 저장하고 관리해야 하는 항목들

- ▶ 데이터를 **입력한 순서대로 저장**하고 꺼낼 수 있음
- ▶ **반복문**을 통해 하나씩 처리 가능
- ▶ **추가, 삭제, 정렬** 등 다양한 조작이 가능함
- ▶ 예: 학생이 오늘 할 일을 저장하고 하나씩 확인하기

리스트 (List)

예시 코드: 데이터 추가 및 참조

```
1 todo = []  
2 todo.append("공부하기")  
3 todo.append("운동하기")  
4 print(todo[0]) # '공부하기'
```

1. todo에 '영화보기' (문자)을 추가해봅시다.
2. todo에 2(숫자)를 추가해봅시다.
3. todo에 False(bool type)를 추가해봅시다.
4. 저장한 값들을 하나씩 확인해봅시다.

딕셔너리 (Dictionary)

실용 예제

단어장, 연락처, 메뉴판

개발 예시

키(key)로 빠르게 찾고 분류해야 하는 데이터

- ▶ "이름 → 전화번호"처럼 하나를 기준으로 값을 찾을 때 유리
- ▶ 검색 속도가 빠르고, 구조가 직관적임
- ▶ 키-값 쌍 (key-value pair)으로 데이터를 저장
- ▶ 예: '김밥'을 검색하면 가격을 바로 출력하는 메뉴판

딕셔너리 (Dictionary)

예시 코드:

```
1 contacts = {  
2     "지민": "010-1234-5678",  
3     "윤기": "010-0000-0000"}  
4 print(contacts["지민"])
```

1. '윤기'의 전화번호를 확인해봅시다.
2. 전화번호의 데이터 type은 무엇일까요?

튜플 (Tuple)

실용 예제

좌석 배치, 좌표 기록

개발 예시

변경되면 안 되는 데이터 쌍 관리

- ▶ **리스트처럼 순서가 있지만, 수정할 수 없음**
- ▶ 고정된 정보 쌍을 안전하게 저장할 때 사용
- ▶ 예: (행, 열) 좌표, (이름, 생년월일) 등
- ▶ 불변성(immutable)을 통해 실수를 방지할 수 있음

튜플 (Tuple)

예시 코드:

```
1 seat = ("B열", 3)
2 print(seat[0])      # 'B열'
3 # seat[1] = 4      # 오류 발생 (튜플은 수정 불가)
```

(응용 예제) 튜플을 하나의 데이터 단위로 하여 리스트에 추가할 수 있음.

```
1 seat_list = []
2 seat_list.append( ("B열", 3) )
3 seat_list.append( ("F열", 5) )
4 seat_list.append( ("A열", 1) )
5 seat_list[1]
```

셋 (Set)

실용 예제

친구 목록 합치기, 투표자 명단 정리

개발 예시

중복 제거와 집합 연산

- ▶ **중복이 자동으로 제거되는 구조**
- ▶ 두 집합의 **합집합, 교집합, 차집합** 연산 가능
- ▶ 예: 중복된 친구 목록을 정리할 때, 투표 중복을 막을 때
- ▶ 순서가 없으므로 정렬이 중요하지 않은 상황에 적합

튜플 (Tuple)

예시 코드:

```
1 a = {"정국", "지민", "태형"}
2 b = {"지민", "윤기", "호석"}
3
4 # 합집합: a 또는 b에 있는 모든 요소
5 print(a | b) # {'정국', '지민', '태형', '윤기', '호석'}
6
7 # 교집합: a와 b 모두에 있는 요소
8 print(a & b) # {'지민'}
9
10 # 차집합: a에는 있지만 b에는 없는 요소
11 print(a - b) # {'정국', '태형'}
```

정리: 어떤 데이터 객체를 써야 할까?

데이터 객체	주요 특징	사용 예시
리스트	순서 있음, 수정 가능	할 일 목록, 쇼핑 리스트
딕셔너리	키-값 구조, 빠른 조회	단어장, 연락처
튜플	순서 있음, 수정 불가	좌표, 고정 데이터
셋	중복 없음, 순서 없음	친구 병합, 투표자 목록

Table: 데이터 객체의 특징과 사용 예시

List

리스트를 배울때 생각해야할 점

- ▶ 리스트를 어떻게 만들어야 하나?
- ▶ 리스트에 어떻게 항목을 추가하나?
- ▶ 리스트에 항목(원소)를 어떻게 확인하나?
- ▶ 리스트의 값을 정렬할 수 있나?
- ▶ 두 개의 리스트를 이어 붙일 수 있나?

리스트

마트에 가서 필요한 물건을 사고자 한다. 필요한 물건들을 리스트(list)로 만들고, 항목을 확인하고, 정리하고 싶다. 이 과정에서 아래와 같은 작업들이 필요하다.

- ▶ 리스트를 어떻게 만들어야 하나?: 처음에 아무것도 없는 상태에서 쇼핑 목록을 만들고 싶음
- ▶ 리스트에 어떻게 항목을 추가하나?: 목록에 '우유'와 '설탕'을 추가하고 싶음
- ▶ 리스트에 항목(원소)를 어떻게 확인하나?: '우유'가 이미 쇼핑 목록에 들어 있는지 확인하고 싶음
- ▶ 리스트에 어떻게 항목을 삭제하나?: 쇼핑 목록에서 설탕을 삭제하고 싶음
- ▶ 리스트의 값을 정렬할 수 있나?: 쇼핑 목록을 가나다순으로 정렬하고 싶음
- ▶ 두 개의 리스트를 이어 붙일 수 있나?: 가족이 만든 별도의 쇼핑 목록이 있을 때 합치고 싶음

리스트를 어떻게 만들어야 하나요?

- ▶ 쇼핑 목록을 비어 있는 리스트로 만들 수 있다.
- ▶ 대괄호 []를 사용한다.
- ▶ 항목은 append 를 이용하여 추가하고 remove 를 이용하여 제거한다.
- ▶ 한번에 여러 개를 만들 수도 있다. []를 사용한다.

예시 코드:

```
1 shopping_list = []  
2 shopping_list.append("우유")  
3 shopping_list.append("사탕")  
4 shopping_list.remove("사탕")  
5 # shopping_list = ['우유', '사탕', '설탕']
```

리스트 항목 확인하기

리스트에 항목이 있는지 확인하려면?

- ▶ "항목" in 리스트 로 확인할 수 있음
- ▶ 예: '우유'가 목록에 있는지 확인
- ▶ 대괄호와 위치 번호를 통해서 확인할 수 있음. (위치 번호는 0부터 시작함. 음수의 경우 역방향으로 인덱스를 참조함.)

예시 코드:

```
1 "우유" in shopping_list
2 shopping_list[2]
3 shopping_list[-1]
```

두 개의 리스트를 합치기

- ▶ + 기호로 리스트를 이어 붙일 수 있음.
- ▶ 가족 쇼핑 목록을 합쳐 하나의 목록 만들기

예시 코드:

```
1 family_list = ["계란", "쌀"]
2 total_list = shopping_list + family_list
3 print(total_list)
```

append() 와 extend() 사용 예시

- ▶ append()는 리스트 하나를 통째로 넣는다.

예시 코드:

```
1 a = [1, 2, 3]
2 b = [4, 5]
3 a.append(b)
4 print(a)
```

- ▶ extend()는 리스트 안의 항목을 하나씩 꺼내서 넣는다. (+ 와 같음)

예시 코드:

```
1 a = [1, 2, 3]
2 b = [4, 5]
3 a.extend(b)
4 print(a)
```

문제 1. 빈 리스트에 항목 추가하기

▶ 아래 코드를 실행했을 때 `fruits` 리스트의 결과는 무엇일까?

```
1 fruits = []  
2 fruits.append("사과")  
3 fruits.append("바나나")  
4 fruits.append("딸기")  
5 print(fruits)
```

문제 2. 항목 삭제 결과 예측하기

▶ 아래 코드를 실행하면 어떤 값이 출력될까요?

```
1 snacks = ["빵", "과자", "젤리"]  
2 snacks.remove("과자")  
3 print(snacks)
```

문제 3. 인덱스로 항목 확인하기

▶ 아래 코드에서 출력되는 값은 무엇인가요?

```
1 colors = ["빨강", "초록", "파랑"]  
2 print(colors[0])  
3 print(colors[-1])
```

문제 4. 리스트 합치기

▶ 아래 코드를 실행하면 total 리스트는 어떤 값을 가질까요?

```
1 morning = ["우유", "식빵"]
2 evening = ["라면", "김치"]
3 total = morning + evening
4 print(total)
```

초급 넘어서기: 인덱스를 이용한 슬라이싱

리스트 슬라이싱 기본

- ▶ 슬라이싱(Slicing): 리스트에서 원하는 부분만 잘라내는 방법
- ▶ 문법: 리스트 [start:end]
- ▶ 시작 인덱스는 포함, 끝 인덱스는 포함하지 않음

예시 코드:

```
1 a = ["가", "나", "다", "라", "마"]  
2 print(a[1:4]) # ["나", "다", "라"]
```

슬라이싱: 생략 가능

- ▶ 시작 또는 끝 인덱스를 생략할 수 있어요.
- ▶ `[:n]` → 처음부터 $n-1$ 까지
- ▶ `[n:]` → n 부터 끝까지

예시 코드:

```
1 a = ["가", "나", "다", "라", "마"]
2 print(a[:3])    # ["가", "나", "다"]
3 print(a[2:])    # ["다", "라", "마"]
```

슬라이싱 확장: step 사용하기

- ▶ 문법: 리스트 [start:end:step]
- ▶ step은 몇 칸씩 건너뛴지를 나타냅니다.

예시 코드:

```
1 a = ["가", "나", "다", "라", "마", "바"]  
2 print(a[1:5:2]) # ["나", "라"]
```

슬라이싱으로 리스트 뒤집기

- ▶ `[::-1]` 을 사용하면 리스트를 뒤집을 수 있어요.
- ▶ step에 음수를 넣으면 거꾸로 진행합니다.

예시 코드:

```
1 a = ["가", "나", "다", "라", "마", "바"]  
2 print(a[::-1]) # ["바", "마", "라", "다", "나", "가"]
```
