Python
CH04: 프로그램의 입출력

전종준 교수

서울시립대학교 통계학과/통계데이터사이언스대학원

Contents

- 오류 처리
- 파일의 읽고 쓰기
- 함수
- 초급 넘어서기: 함수 매개변수의 순서

오류 처리

예외 처리는 왜 필요한가?

- 사용자 입력을 여러 번 받고, 각각 10을 나눠 출력
- 잘못된 입력이 있으면 반복문이 멈춤
- 오류가 발생하는 경우 메세지를 발생시키지만 반복은 계속되도록 할 수 있을까?

```
1  for i in range(3):
2      num = int(input("숫자 입력: "))
3      print("10 /", num, "=", 10 / num)
```

예외 처리: try와 except

- 프로그램 실행 중 오류가 발생할 수 있는 부분을 try로 감싼다
- 오류가 발생하면 except 블록으로 이동
- 오류의 종류를 확인하고 이어지는 문장을 실행함.

예시 코드:

```
1 try:
2     x = int(input("숫자를 입력하세요: "))
3     print(10 / x)
4 except ZeroDivisionError:
5     print("0으로 나눌 수 없습니다.")
6 except ValueError:
7     print("숫자를 입력해주세요.")
```

대표 오류 예시들

- `ZeroDivisionError`: 0으로 나눌 때
- `ValueError`: 부적절한 값 사용
- `TypeError`: 자료형이 잘못되었을 때
- `IndexError`: 리스트 등에서 잘못된 인덱스 접근
- `KeyError`: 딕셔너리에 존재하지 않는 키 접근
- `NameError`: 정의되지 않은 변수 사용
- `ImportError`: 모듈 가져오기 실패

예시 코드:

```
1  # ZeroDivisionError
2  print(10 / 0)
3  # ValueError
4  int("hello")
5  # TypeError
6  print("문자열" + 123)
7  # IndexError
8  lst = [1, 2, 3]
9  print(lst[5])
10 # KeyError
11 info = {"name": "Tom"}
12 print(info["age"])
13 # NameError
14 print(x2140)
15 # ImportError
16 import non_existent_module
```

finally: 항상 실행되는 블록

- try 블록에서 예외가 발생하든 안 하든 finally는 무조건 실행됩니다.
- 정리 작업(예: 연결 종료, 메시지 출력 등)에 자주 사용됩니다.

예시 코드:

```
try:
    x = int(input("숫자를 입력하세요: "))
    result = 10 / x
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
finally:
    print("프로그램 종료")
```

raise: 예외를 강제로 발생시키기

- raise 문을 사용하면 원하는 시점에 예외를 발생시킬 수 있습니다.
- 유효성 검사나 조건 검출에 활용됩니다.

예시 코드:

```
age = int(input("나이를 입력하세요: "))
if age < 0:
    raise ValueError("나이는 0보다 작을 수 없습니다.")
print("나이:", age)
```

예제 1: 반복문 안에서의 오류 처리

- 사용자로부터 여러 값을 입력받을 때,
- 하나의 입력에서 오류가 나도 전체 루프가 멈추지 않게 하려면
- `try-except`를 반복문 안에 넣어야 합니다.

```
# 잘못된 형식의 입력이 발생하는 경우에도 반복문이 실행됨
for i in range(3):
    try:
        num = int(input("숫자 입력: "))
        print("10 /", num, "=", 10 / num)
    except ValueError:
        print("숫자가 아닙니다.")
    except ZeroDivisionError:
        print("0으로 나눌 수 없습니다.")
```

예제 2: 리스트 요소를 숫자로 변환

- 문자열 리스트를 숫자로 바꾸며 합계 계산
- 변환할 수 없는 항목은 건너뜀

```
data = ["10", "20", "hello", "30"]
total = 0

for item in data:
    try:
        total += int(item)
    except ValueError:
        print(f"변환 실패: {item}")

print("합계:", total)
```

파일의 읽고 쓰기

문자열 포매팅이란?

- 문자열 안에 변수나 값을 삽입하는 방법
- 다양한 방식으로 데이터를 보기 좋게 출력할 수 있음
- `f-string`, `format()`, `%` 연산자 방식이 대표적임.

f-string 사용하기

- 문자열 앞에 `f`를 붙이고, 중괄호 `{}` 안에 변수 사용

예시 코드:

```
1 name = "지민"
2 age = 15
3 print(f"{name}은 {age}살입니다.")
```

format() 메서드 사용하기

- {} 안에 값이 들어감.
- 순서대로 삽입하거나, 인덱스/이름 지정도 가능.

예시 코드:

```
1 print("{}은 {}살입니다.".format("지민", 15))
2 print("{1}살인 {0}입니다.".format("지민", 15))
3 print("{name}은 {age}살입니다.".format(name="지민", age=15))
```

형식 지정자 정리

- 숫자, 문자열을 다양한 방식으로 정렬하거나 표현 가능
- `f-string`, `format()` 둘 다 사용 가능
- {문자열:형식지정자} 또는 {숫자:형식지정자} 또는 {변수:형식지정자} 로 사용함. 예를 들어 {price:,.2f} 는 price 가 변수이고 변수값을 .2f 인 형식으로 표시

예시 코드:

```
1 price = 1234.567
2 print(f"가격: {price:,.2f}원")  # 천 단위 쉼표, 소수점 둘째 자리
3 print(f"{'Python':^10}")       # 가운데 정렬
4 print(f"{42:05}")              # 다섯 자리, 빈칸 0으로 채움
```

format()과 형식 지정자

- `.format()` 메서드에서도 `:`형식을 사용할 수 있어요.
- 콤마, 소수점 자릿수, 정렬 등을 지정할 수 있어요.

예시 코드:

```
price = 9876543.21
name = "지민"

print("이름: {name:<10} | 금액: {price:>15,.2f}원".format(name=name, price=price))
```

(확인)

- :<10 왼쪽 정렬, 10칸 확보
- :>15,.2f 오른쪽 정렬, 15칸 확보, 천단위 쉼표, 소수점 둘째 자리까지 실수

파일 입출력이란?

- 파이썬 코드에서 파일을 열어 데이터를 읽거나 쓸 수 있음
- 대표 함수: `open()`, `read()`, `write()`, `close()`
- 텍스트 파일이 가장 기본적인 입출력 대상임

텍스트 파일 쓰기

- `open()` 함수로 파일을 열고, `write()`로 내용 작성
- “w“ 모드는 기존 내용을 덮어 씀

예시 코드:

```
1 f = open("memo.txt", "w")
2 f.write("파이썬 파일 쓰기 연습 중입니다.\n")
3 f.write("한 줄 더 써볼게요.")
4 f.close()
```

텍스트 파일 읽기

- `read()` 또는 `readlines()`를 사용
- 파일은 `open()` 후 꼭 `close()` 해야 함. (File connection을 끊는다는 의미)

예시 코드:

```
1 f = open("memo.txt", "r")
2 data = f.read()
3 print(data)
4 f.close()
```

파일에 이어 쓰기

- “a“ 모드로 열면 기존 내용을 지우지 않고 덧붙일 수 있어요.

예시 코드:

```
f = open("memo.txt", "a")
f.write("프로그램 실행됨\n")
f.write("사용자 입력 기록됨\n")
f.close()
```

with 문으로 안전하게 파일 다루기

- `with open()` 구문을 사용하면 자동으로 파일을 닫아줘요.

예시 코드:

```
with open("memo.txt", "r") as f:
    f.write("안전한 사용!\n")
```

실습: 메모 저장기 만들기

- 사용자로부터 입력을 받아 `memo.txt`에 저장해보세요.
- 파일을 열어 내용을 다시 출력해보세요.

힌트:

```
memo = input("메모를 입력하세요: ")
with open("memo.txt", "w") as f:
    f.write(memo)
```

(주의) w 옵션은 memo.txt를 새로 씁니다.

함수

함수의 정의

- 반복되는 코드를 묶는 기능
- 입력값을 받아 결과값을 반환하는 구조
- 프로그램의 구조화와 재사용성 향상

함수의 선언 방법

- `def` 키워드를 사용하는 선언문
- 소괄호 안에 매개변수 작성
- 콜론(:) 뒤에 들여쓴 코드 블록 작성
- 코드 블록은 indent 필요함

예시 코드:

```
1 def say_hello():
2     print("안녕하세요!")
```

함수의 호출

- 함수 이름 뒤에 소괄호를 붙여 실행
- 인자가 필요한 경우 괄호 안에 전달
- 정의된 함수 본문의 코드 실행

예시 코드:

```
1 say_hello()
```

매개변수와 인자

- 매개변수: 함수 정의 시 사용하는 변수
- 인자: 함수 호출 시 전달하는 값
- 입력값을 받아 동적으로 작동하는 함수 구현

예시 코드:

```
1 def greet(name):
2     print(f"{name}님, 환영합니다!")
3
4 greet("철수")
```

반환값

- `return` 키워드를 사용하는 반환문
- 결과값을 함수 밖으로 전달
- 다양한 계산과 처리를 함수 내부에서 수행 가능

예시 코드:

```
1 def add(a, b):
2     return a + b
3
4 result = add(3, 5)
5 print(result)
```

기본값 매개변수

- 인자가 전달되지 않아도 작동하는 설정
- 함수 선언 시 = 기호로 기본값 지정
- 호출 시 일부 인자 생략 가능

예시 코드:

```
1 def greet(name="손님"):
2     print(f"{name}님, 안녕하세요!")
3
4 greet()
```

연습문제 1: 위치 매개변수

- 이름과 나이를 받아 출력하는 함수 `introduce()`를 작성
- 출력 예: 홍길동은 30살입니다.
- 함수 호출 예: `introduce("홍길동", 30)`

연습문제 2: 기본값 매개변수

- 이름을 받아 인사하는 함수 `greet()`를 작성하세요.
- 인사가 기본값으로 "안녕하세요"를 사용하도록 하세요.
- 출력 예: 수지님, 안녕하세요!
- 호출 예: `greet("수지")` 또는 `greet("민수", "반가워요")`

초급 넘어서기: 함수 매개변수의 순서

매개변수와 인자의 개념

- **매개변수(Parameter)**: 함수 정의 시 사용하는 변수
- **인자(Argument)**: 함수 호출 시 실제로 전달하는 값
- 함수는 매개변수를 통해 외부 입력을 받아 처리

예시 설명:

```
1  def greet(name):      # name → 매개변수
2      print(f"{name}님, 안녕하세요!")
3
4  greet("영희")          # "영희" → 인자
```

매개변수의 종류

- **위치 매개변수 (positional parameter)**: 순서대로 인자를 받음
- **기본값 매개변수 (default parameter)**: 인자가 없을 때 기본값 사용
- **가변 위치 매개변수 (*args)**: 여러 개의 위치 인자 수용 (튜플)
- **가변 키워드 매개변수 (**kwargs)**: 여러 개의 키워드 인자 수용 (딕셔너리)
- **키워드 전용 매개변수**: 키워드로만 전달 가능한 매개변수

위치 vs 기본값 매개변수

- 위치 매개변수는 순서대로 값을 전달받음
- 기본값 매개변수는 값이 생략될 경우 기본값 사용

예시 코드:

```
1  def introduce(name, age=20):
2      print(f"{name}은 {age}살입니다.")
3
4  introduce("영희")          # age는 기본값 사용
5  introduce("철수", 25)       # age는 25로 대체
```

키워드 전용 매개변수

- `*` 이후에 오는 매개변수는 키워드로만 전달 가능
- 위치 인자로 전달할 수 없음
- 명확한 인자 지정이 필요할 때 사용

예시 코드:

```
1  def draw_circle(radius, *, color="black"):
2      print(f"{color} 원, 반지름: {radius}")
3
4  draw_circle(5, color="red")     # OK
5  draw_circle(5, "red")           # 오류 발생
```

키워드 인자

- 인자의 이름을 명시하여 전달
- 순서에 관계없이 명확한 의미 전달 가능
- 가독성과 유지보수성 향상

예시 코드:

```
1 def introduce(name, age):
2     print(f"{name}은 {age}살입니다.")
3
4 introduce(age=20, name="영희")
```

가변 위치 인자 (*args)

- 개수 제한 없이 위치 인자 전달
- 튜플 형태로 함수 내부에서 처리
- 반복적 입력이 필요한 경우 유용

예시 코드:

```
1  def total(*numbers):
2      print(sum(numbers))
3
4  total(1, 2, 3, 4)
```

*args와 반복문

- 여러 숫자를 받아 하나씩 출력하는 함수
- 반복문을 사용해 인자를 순회 처리

예시 코드:

```
1  def print_numbers(*args):
2      for num in args:
3          print(f"숫자: {num}")
4
5  print_numbers(3, 7, 10)
```

가변 키워드 인자 (**kwargs)

- 이름이 붙은 인자를 딕셔너리 형태로 전달
- 키워드 인자의 집합 처리 가능
- 옵션 설정이나 조건 분기 처리에 유용

예시 코드:

```python
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="민수", age=22)
```

연습문제 1: `*args` 활용

- 여러 개의 숫자를 받아 합계를 출력하는 함수 `total()`를 작성하세요.
- 인자의 개수는 정해지지 않음
- 출력 예: `총합: 15`
- 호출 예: `total(1, 2, 3, 4, 5)`

연습문제 2: **kwargs 활용

- 사람의 정보를 받아 출력하는 함수 `print_info()`를 작성하세요.
- 키워드 인자를 받아 한 줄씩 출력
- 호출 예: `print_info(name="수지", age=25, city="서울")`
- 출력 예:
  - name: 수지
  - age: 25
  - city: 서울