





오늘의 학습 목표

-  다양한 파일 형식을 자유자재로 다루기
-  예외 처리로 견고한 프로그램 만들기
-  정규표현식으로 텍스트 처리하기
-  효과적인 디버깅과 로깅 기법 익히기

수업 구성

1. 파일 입출력과 데이터 처리
2. 예외 처리와 디버깅 기법
3. 정규표현식과 텍스트 처리
4. 다음 주 미리보기 & 과제

1부: 파일 입출력과 데이터 처리

파일 입출력 심화 - with문

❌ 권장하지 않는 방법

```
file = open('data.txt', 'r')  
content = file.read()  
file.close() # 잊기 쉬움!
```

✅ 권장하는 방법

```
with open('data.txt', 'r') as file:  
    content = file.read()  
# 자동으로 close() 호출됨
```

텍스트 파일 vs 바이너리 파일

```
# 텍스트 파일
with open('text.txt', 'r', encoding='utf-8') as f:
    text = f.read()

# 바이너리 파일
with open('image.png', 'rb') as f:
    binary_data = f.read()
```

주의: 이미지, 동영상, 압축파일은 반드시 바이너리 모드로!

pathlib 모듈로 경로 다루기

```
from pathlib import Path

# 경로 생성
path = Path('data') / 'files' / 'test.txt'

# 유용한 메서드들
if path.exists():
    print(f"파일 크기: {path.stat().st_size} bytes")
    print(f"확장자: {path.suffix}")
    print(f"부모 폴더: {path.parent}")
```

대용량 파일 효율적으로 읽기

❌ 메모리 부족 위험

```
content = open('huge_file.txt').read()
```

✅ 청크 단위로 읽기

```
def read_large_file(file_path, chunk_size=1024):  
    with open(file_path, 'r') as file:  
        while True:  
            chunk = file.read(chunk_size)  
            if not chunk:  
                break  
            yield chunk
```

파일 포인터 활용

```
with open('data.txt', 'r') as f:
    # 현재 위치 확인
    print(f.tell()) # 0

    # 10번째 바이트로 이동
    f.seek(10)

    # 현재 위치부터 읽기
    content = f.read(5)
```


파일 모드 완벽 정리

모드	설명	파일 없으면
r	읽기 전용	에러 발생
w	쓰기 (덮어쓰기)	새로 생성
a	추가 쓰기	새로 생성
r+	읽기/쓰기	에러 발생
w+	읽기/쓰기 (초기화)	새로 생성
rb , wb	바이너리 모드	-

CSV 파일 다루기

```
import csv

# CSV 읽기
with open('students.csv', 'r', encoding='utf-8') as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(f"{row['이름']}: {row['점수']}점")

# CSV 쓰기
with open('output.csv', 'w', newline='', encoding='utf-8') as f:
    writer = csv.DictWriter(f, fieldnames=['이름', '점수'])
    writer.writeheader()
    writer.writerow({'이름': '김철수', '점수': 95})
```

JSON 데이터 처리

```
import json

# JSON 읽기
with open('config.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

# JSON 쓰기 (한글 처리)
with open('output.json', 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=2)
```

파일 인코딩 이슈 해결

```
# 인코딩 자동 감지
import chardet

with open('unknown.txt', 'rb') as f:
    result = chardet.detect(f.read())
    encoding = result['encoding']

# 올바른 인코딩으로 읽기
with open('unknown.txt', 'r', encoding=encoding) as f:
    content = f.read()
```

pickle로 Python 객체 저장

```
import pickle

# 객체 저장
data = {'name': '홍길동', 'scores': [90, 85, 88]}
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)

# 객체 불러오기
with open('data.pkl', 'rb') as f:
    loaded_data = pickle.load(f)
```

설정 파일 다루기

```
import configparser

config = configparser.ConfigParser()
config.read('settings.ini', encoding='utf-8')

# 설정 읽기
db_host = config['DATABASE']['host']
db_port = config.getint('DATABASE', 'port')

# 설정 쓰기
config['DATABASE']['password'] = 'new_password'
with open('settings.ini', 'w') as f:
    config.write(f)
```

os와 shutil 모듈 활용

```
import os
import shutil

# 파일 작업
os.rename('old.txt', 'new.txt')
shutil.copy('source.txt', 'destination.txt')
shutil.move('file.txt', 'folder/')

# 디렉토리 작업
os.makedirs('new/folder/path', exist_ok=True)
shutil.rmtree('folder_to_delete')
```

파일 메타데이터 확인

```
import os
from datetime import datetime

stat = os.stat('file.txt')

print(f"크기: {stat.st_size:,} bytes")
print(f"수정 시간: {datetime.fromtimestamp(stat.st_mtime)}")
print(f"권한: {oct(stat.st_mode)[-3:]}")
```


디렉토리 순회하기

```
import os
import glob

# os.walk 사용
for root, dirs, files in os.walk('project'):
    for file in files:
        if file.endswith('.py'):
            print(os.path.join(root, file))

# glob 패턴 사용
for file in glob.glob('**/*.txt', recursive=True):
    print(file)
```

실습 1: 로그 분석기

웹서버 로그 파일을 분석하는 프로그램을 만들어봅시다!

```
192.168.1.1 - - [10/Oct/2024:13:55:36] "GET /index.html" 200
192.168.1.2 - - [10/Oct/2024:13:56:12] "POST /login" 404
```

구현할 기능:

- IP별 접속 횟수 집계
- 시간대별 트래픽 분석
- 에러 코드별 통계

실습 2: 스마트 백업 프로그램

변경된 파일만 백업하는 똑똑한 프로그램을 만들어봅시다!

구현할 기능:

- 마지막 백업 이후 변경된 파일 감지
- 백업 히스토리 관리
- ZIP 압축 백업 옵션

2부: 예외 처리와 디버깅 기법

try-except-else-finally 구조

```
try:
    # 위험한 코드
    result = 10 / x
except ZeroDivisionError:
    # 예외 처리
    print("0으로 나눌 수 없습니다!")
except Exception as e:
    # 다른 예외들
    print(f"에러 발생: {e}")
else:
    # 예외가 없을 때만 실행
    print(f"결과: {result}")
finally:
    # 무조건 실행
    print("계산 완료")
```

여러 예외 동시에 처리

```
try:
    value = int(input("숫자 입력: "))
    result = 10 / value
except (ValueError, ZeroDivisionError) as e:
    print(f"입력 오류: {e}")
```

예외별 다른 처리

```
try:
    # 코드
except ValueError:
    print("숫자가 아닙니다")
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다")
```

예외 체이닝

```
def process_data(filename):  
    try:  
        with open(filename) as f:  
            data = json.load(f)  
    except FileNotFoundError as e:  
        raise ValueError(f"설정 파일을 찾을 수 없습니다") from e  
    except json.JSONDecodeError as e:  
        raise ValueError(f"잘못된 JSON 형식") from e
```

사용자 정의 예외 클래스

```
class ValidationError(Exception):  
    """데이터 검증 실패"""  
    pass  
  
class AuthenticationError(Exception):  
    """인증 실패"""  
    def __init__(self, username):  
        self.username = username  
        super().__init__(f"인증 실패: {username}")  
  
# 사용 예시  
if not password_correct:  
    raise AuthenticationError(username)
```


컨텍스트 관리자 만들기

```
class DatabaseConnection:
    def __enter__(self):
        print("DB 연결")
        self.connection = create_connection()
        return self.connection

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("DB 연결 종료")
        self.connection.close()
        return False # 예외 전파

# 사용
with DatabaseConnection() as db:
    db.execute("SELECT * FROM users")
```

예외 처리 모범 사례

✓ DO

- 구체적인 예외 타입 사용
- 예외 메시지 명확하게 작성
- 로깅과 함께 사용

✗ DON'T

- 모든 예외를 잡는 bare except 사용
- 예외 무시하기 (pass만 쓰기)
- 너무 큰 try 블록

Python 디버거(pdb) 사용법

```
import pdb

def complex_function(x, y):
    result = x * y
    pdb.set_trace() # 여기서 멈춤
    result = result + 10
    return result
```

```
# 디버거 명령어
# n: 다음 줄
# s: 함수 내부로
# c: 계속 실행
# p 변수명: 변수 값 출력
```

logging 모듈 기초

```
import logging

# 로거 설정
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('app.log'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)
```

로그 레벨과 사용법

```
# 로그 레벨 (낮음 → 높음)
logger.debug("디버깅 정보")
logger.info("일반 정보")
logger.warning("경고 메시지")
logger.error("에러 발생")
logger.critical("심각한 문제")

# 예외 로깅
try:
    risky_operation()
except Exception as e:
    logger.error("작업 실패", exc_info=True)
```

파일 로테이션

```
from logging.handlers import RotatingFileHandler

# 파일 크기 기반 로테이션
handler = RotatingFileHandler(
    'app.log',
    maxBytes=10*1024*1024, # 10MB
    backupCount=5
)

# 시간 기반 로테이션
from logging.handlers import TimedRotatingFileHandler
handler = TimedRotatingFileHandler(
    'app.log',
    when='midnight',
    interval=1,
    backupCount=30
)
```

성능 프로파일링

```
import time
import timeit

# 간단한 시간 측정
start = time.time()
# 코드 실행
end = time.time()
print(f"실행 시간: {end - start:.2f}초")

# timeit으로 정밀 측정
result = timeit.timeit(
    'sum([i**2 for i in range(1000)])',
    number=10000
)
print(f"평균 실행 시간: {result/10000:.6f}초")
```

실습: 견고한 주소록 앱

요구사항:

- CSV/JSON 형식 지원
- 데이터 검증 (이메일, 전화번호)
- 자동 백업 기능
- 상세한 로깅
- 검색 및 정렬

예외 처리 포인트:

- 파일 읽기/쓰기 오류
- 잘못된 데이터 형식
- 중복 데이터

3부: 정규표현식과 텍스트 처리

정규표현식 기초

```
import re

# 기본 패턴 매칭
pattern = r'python'
text = "I love Python programming"

# 대소문자 무시
match = re.search(pattern, text, re.IGNORECASE)
if match:
    print(f"찾음: {match.group()}")
```

주요 메타문자

패턴	의미	예시
.	임의의 한 문자	<code>a.c</code> → abc, a1c
*	0회 이상 반복	<code>ab*c</code> → ac, abc, abbc
+	1회 이상 반복	<code>ab+c</code> → abc, abbc
?	0회 또는 1회	<code>ab?c</code> → ac, abc
[]	문자 클래스	<code>[a-z]</code> → 소문자
^	문자열 시작	<code>^Hello</code>
\$	문자열 끝	<code>world\$</code>

그룹과 캡처

```
# 그룹 사용
pattern = r'(\d{3})-(\d{4})-(\d{4})'
phone = "010-1234-5678"

match = re.match(pattern, phone)
if match:
    print(f"전체: {match.group(0)}")
    print(f"앞자리: {match.group(1)}")
    print(f"중간: {match.group(2)}")
    print(f"끝자리: {match.group(3)}")
```

실용적인 패턴들

이메일

```
email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
```

한국 전화번호

```
phone_pattern = r'^01[0-9]-?\d{3,4}-?\d{4}$'
```

URL

```
url_pattern = r'https?://(?:www\.)?[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
```

IP 주소

```
ip_pattern = r'^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$'
```

주요 메서드들

```
# search: 첫 번째 매치 찾기  
re.search(pattern, text)
```

```
# match: 문자열 시작부터 매치  
re.match(pattern, text)
```

```
# findall: 모든 매치 찾기  
emails = re.findall(email_pattern, text)
```

```
# sub: 치환  
cleaned = re.sub(r'^\w\s', '', text) # 특수문자 제거
```

실습: 텍스트 데이터 클리닝

```
# 로그에서 IP 추출
log = "192.168.1.1 - - [10/Oct/2024:13:55:36]"
ip = re.search(r'\d+\.\d+\.\d+\.\d+', log)

# 민감정보 마스킹
text = "연락처: 010-1234-5678, 주민번호: 901010-1234567"
masked = re.sub(r'\d{6}-\d{7}', '*****-*****', text)

# 텍스트 정규화
text = "파이썬Python 프로그래밍Programming"
normalized = re.sub(r'\s+', ' ', text) # 연속 공백 제거
```

4부: 다음 주 내용 미리보기

3주차: 객체지향 프로그래밍 심화

- 클래스 상속과 다형성
 - 부모/자식 클래스
 - 메서드 오버라이딩
 - `super()` 활용
- 특수 메서드 (매직 메서드)
 - `__str__` , `__repr__`
 - `__len__` , `__getitem__`
 - `__add__` , `__eq__`

3주차: 고급 OOP 기법

- 프로퍼티와 디스크립터

```
@property
def age(self):
    return self._age

@age.setter
def age(self, value):
    if value < 0:
        raise ValueError("나이는 음수일 수 없습니다")
    self._age = value
```

4주차: 웹 스크래핑 기초

- requests로 웹 데이터 가져오기
- BeautifulSoup으로 HTML 파싱
- API 활용하기
 - REST API 이해
 - JSON 응답 처리
 - 인증과 헤더

실습: 날씨 정보 수집기, 뉴스 헤드라인 스크래퍼





5주차: 데이터 분석 입문

- pandas 기초
 - DataFrame 다루기
 - 데이터 필터링과 정렬
 - 그룹별 집계
- 데이터 시각화
 - matplotlib 기본
 - seaborn으로 통계 플롯

실습: 실제 데이터셋 분석 프로젝트

마무리

오늘 배운 내용

-  다양한 파일 형식 처리
-  견고한 예외 처리
-  정규표현식 기초
-  로깅과 디버깅

다음 시간

- 객체지향 프로그래밍 심화

질문 있으신가요? 🤔

수고하셨습니다! 🙌