Exception Handling & File I/O

ESM2014-41

객체지향프로그래밍 및 실습

SKKU 시스템경영공학과

조영일

Syntax Error vs Exception

- Syntax Error : 문법 상의 오류로 실행 시점 이전에 발견되는 코드 작성 상의 문제점. 물론 코드 실행도 불가하다.(잘못 작성된 코드에서 기인함)

- Exception(예외) : 문법 상의 오류는 없지만, 실행 시점에 발견되는 프로그램 상의 오류.(정상 코드에서도 발생 할 수 있음)

Exception Handling

- Exception Handling(예외 처리) : Exception(예외)가 발생할 것으로 예측되는 지점에서, Exception에 대한 적절한 처리 방법을 작성하여 프로그램의 정상 동작을 보장하는 것

- Exception Handling 하지 않은 상태에서 Exception 이 발생 할 경우?
  Exception 발생 시점 부터 프로그램의 실행이 중단됨

try, except

- try, except 문 : 예외가 발생 하였을 때 적절한 처리 방법을 직접 지정함

```
try:
    # 예외가 발생 할 것으로 예상되는 Code Block
    x = 4/0
except:
    # 예외 처리
    print("Zero Division Exception!")
```

try, except

- except 만을 사용해서 모든 예외를 하나의 code block 에서 처리하는 것은 좋지 않다.
- except [예외의 종류] 구문을 사용해서 서로 다른 예외에 대한 처리 방법을 명시적으로 작성한다.

```
try:
    # 예외가 발생 할 것으로 예상되는 Code Block
    x = value_list[0]/value_list[3]
except ZeroDivisionError:
    # value_list[3]이 0인 경우에 대한 예외 처리
    print("Zero Division Exception!")
except IndexError:
    # value_list에 존재하지 않는 index를 탐색하려 할 때에 대한 예외 처리
    print("List Index Exception!")
```

Built-in exception list

- Python에서 빈번하게 발생 할 수 있는 예외 상황에 대한 정의는 기본적으로 Python에 Built-in exception 으로 포함되어 있음

- IndexError : 연속된 시퀀스(ex. List)를 index로 탐색 할 때 index가 시퀀스의 범위를 벗어날 때 발생
- KeyError : Dictionary에서 key로 탐색을 할 때 해당 key가 존재하지 않을 때 발생
- ZeroDivisionError : 0으로 나누려 할 때 발생

- https://docs.python.org/ko/3/library/exceptions.html#

try, except

- except [예외의 종류] as [변수명] : 예외의 구체적인 내용(메시지)를 변수에 할당하여 except 코드 블록 안에서 사용하고자 할때

```
try:
    # 예외가 발생 할 것으로 예상되는 Code Block
    x = value_list[0]/value_list[3]
except ZeroDivisionError as x:
    # value_list[3]이 0인 경우에 대한 예외 처리
    print("Error : " + x)
```

try, except, finally

- finally : try 문과 연달아 사용 할 경우, try 문 수행 도중 예외 발생 여부와 상관 없이 코드 블록 실행

```
try:
    # 예외가 발생 할 것으로 예상되는 Code Block
    x = value_list[0]/value_list[3]
except ZeroDivisionError as x:
    # value_list[3]이 0인 경우에 대한 예외 처리
    print("Error : " + x)
finally:
    print("End of try code block execution")
```

raise

- raise : 내가 작성한 코드에서 특정한 상황에 예외를 임의로 발생 시키고 싶을 경우

```
class Student:
    def study(self):
        raise NotImplementedError


s = Student()
s.study()
```

```
Traceback (most recent call last):
  File "C:\Users\jeffr\.IntelliJIdea2019.1\config\plugins\python\helpers\pydev\pydevd.py", line 1758, in <module>
    main()
  File "C:\Users\jeffr\.IntelliJIdea2019.1\config\plugins\python\helpers\pydev\pydevd.py", line 1752, in main
    globals = debugger.run(setup['file'], None, None, is_module)
  File "C:\Users\jeffr\.IntelliJIdea2019.1\config\plugins\python\helpers\pydev\pydevd.py", line 1147, in run
    pydev_imports.execfile(file, globals, locals)  # execute the script
  File "D:/skku.py", line 189, in <module>
    s.study()
  File "D:/skku.py", line 185, in study
    raise NotImplementedError
NotImplementedError
```

File I/O

- Console I/O : Console 을 통한 입력과 출력
- File I/O : File 을 통한 입력과 출력

open()/close()/write() method

- open() : 파일 내용을 읽거나 쓰기 위하여 여는 built-in function
- 변수명 = open(“파일 경로”, “파일 열기 모드”)

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
w	쓰기모드 - 파일에 내용을 쓸 때 사용(항상 파일 내용을 초기화 시킨다.)
a	추가모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용

open()/close()/write() method

- close() : 작업이 끝난 파일 객체를 닫는 method
  - open() 할 경우 파일의 내용이 컴퓨터 메모리에 적재됨. 사용하지 않는 파일일 경우 닫아줘야 메모리 부족이나, 파일 충돌 등의 문제를 방지 할 수 있음

- write() : 파일에 내용을 기록하기 위해 사용하는 method

open()/close()/write() method

```
# 파일을 쓰기 모드로 연다.
f = open("new_file_1.txt", 'w')
# 파일 객체를 닫는다.
f.close()

# 파일을 쓰기 모드로 연다.
f = open("new_file_2.txt", 'w')
for i in range(1, 11):
    data = "%d line\n" % i
    f.write(data)
f.close()
```

readline(), readlines(), read() method

- readline() : 파일의 내용을 한 줄씩 반환한다.
- readlines() : 파일의 내용을 모두 읽고, 각 줄을 원소로 하는 list를 반환한다.
- read(size) : size(bytes) 만큼 파일의 내용을 읽고 그 내용을 반환함. size를 전달 하지 않을 경우 파일 전체 내용을 반환

readline(), readlines(), read() method

- E.g. - file.txt

line1₩n
line2₩n
line3₩n
line4₩n
line5₩n
line6₩n

readline(), readlines(), read() method

```
f = open("file.txt", 'r')
while True:
    line = f.readline()
    if not line: break
    print(line)
f.close()


f = open("file.txt", 'r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()


f = open("file.txt", 'r')
file_content = f.read()
print(file_content)
f.close()
```

Summary

- Syntax Error vs Exception
- try, except, finally, raise
- File I/O