

Chapter12 – 예외 처리와 파일

예외(Exception)

1) 예상이 가능한 예외

-발생여부를 사전에 인지할 수 있는 예외(잘못된 입력, 파일 호출시 파일 없음)

-개발자가 반드시 명시적으로 정의 해야 함

2) 예상이 불가능한 예외

-인터프리터 과정에서 발생하는 예외, 개발자 실수

예외 처리(Exception Handling)

-예외가 발생할 경우 후속 조치 등 대처 필요

1) 없는 파일 호출 -> 파일 없음을 알림

2) 게임 이상 종료 -> 게임 정보 저장

try ~ except 문법

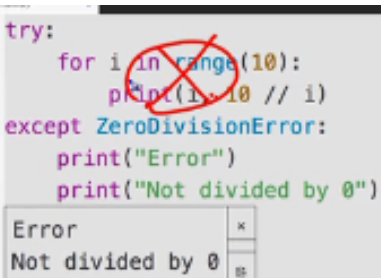
```
try:  
    예외 발생 가능 코드  
except <Exception Type>:  
    예외 발생시 대응하는 코드
```

- 0으로 숫자를 나눌 때 예외처리 하기

```
for i in range(10):  
    try:  
        print(10 / i)  
    except ZeroDivisionError:  
        print("Not divided by 0")
```

*try구문이 이후 맞다면 정상작동, 그렇지 않다면 스킵하게된다.

```
****  
try:  
    for i in range(10):  
        print(10 // i)  
except ZeroDivisionError:  
    print("Error")  
    print("Not divided by 0")
```



try ~ except ~ finally

try:

예외 발생 가능 코드

except <Exception Type>:

예외 발생시 동작하는 코드

finally:

예외 발생 여부와 상관없이 실행됨

```
try:
    for i in range(0, 10):
        result = 10 // i
        print(i, "-----", result)
except ZeroDivisionError:
    print("Not divided by 0")
finally:
    print("종료되었습니다.")
```

Not divided by 0
종료되었습니다.

-Built-in Exception

IndexError : List의 index 범위를 넘어갈 때

NameError : 존재하지 않은 변수를 호출 할 때

ZeroDivisionError : 0으로 숫자를 나눌 때

Value Error : 변환할 수 없는 문자/숫자를 변환할 때

fileNotFoundError : 존재하지 않는 파일을 호출할 때

raise구문 - 필요에 따라 강제로 Exception을 발생

*raise <Exception Type>(예외정보)

assert구문 - 특정 조건에 만족하지 않을 경우 예외 발생

*assert 예외조건

File Handling Overview

파일의 종류

Binary 파일	Text 파일
- 컴퓨터만 이해할 수 있는 형태인 이진(법)형식 으로 저장된 파일	- 인간도 이해할 수 있는 형태인 문자열 형식 으로 저장된 파일
- 일반적으로 메모장으로 열면 내용이 깨져 보임 (메모장 해설 불가)	- 메모장으로 열면 내용 확인 가능
- 엑셀파일, 워드 파일 등등	- 메모장에 저장된 파일, HTML 파일, 파이썬 코드 파일 등

- 컴퓨터는 Text 파일을 처리하기 위해 Binary 파일로 변환시킴 (예: pyc 파일)
- 모든 Text 파일도 실제로는 Binary 파일, ASCII/Unicode 문자열 집합으로 저장되어 사람이 읽을 수 있음

File Handling1

File read

파이썬은 파일 처리를 위해 "open" 키워드를 사용함

```
f = open("<파일이름>", "접근 모드")  
f.close()
```

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
w	쓰기모드 - 파일에 내용을 쓸 때 사용
a	추가모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용

-open명령어는 인터프리터가 하드에 있는 자료와 연결시켜주는 역할을 하는 것이다.

-read()명령어를 따로 입력해서 안쪽에 컨텐츠들을 가져와준다.

-파일을 사용하고 있으면 삭제가 되지 않기에 close명령어를 실행 후 삭제를 한다.

read() txt 파일 안에 있는 내용을 문자열로 반환

```
f = open("i_have_a_dream.txt", "r" )  
contents = f.read()  
print(contents)  
f.close()
```

대상파일이 같은 폴더에 있을 경우

with 구문과 함께 사용하기

```
with open("i_have_a_dream.txt", "r") as my_file:  
    contents = my_file.read()  
    print (type(contents), contents)
```

한 줄씩 읽어 List Type으로 반환함

```
with open("i_have_a_dream.txt", "r") as my_file:  
    content_list = my_file.readlines() #파일 전체를 list로 반환  
    print(type(content_list)) #Type 확인  
    print(content_list) #리스트 값 출력
```

실행 시 마다 한 줄 씩 읽어 오기

```
with open("i_have_a_dream.txt", "r") as my_file:  
    i = 0  
    while 1:  
        line = my_file.readline()  
        if not line:  
            break  
        print (str(i) + " == " + line.replace("\n","")) #한줄씩 값 출력  
        i = i + 1
```

File write

```
f = open("count_log.txt", 'w', encoding="utf8")
for i in range(1, 11):
    data = "유원상 %d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

-open과 유사한 과정으로 파일을 불러 온 다음 write명령어를 통해 파일을 읽어준다.

```
with open("count_log.txt", 'a', encoding="utf8") as f:
    for i in range(100, 111):
        data = "%d번째 줄입니다.\n" % i
        f.write(data)
```

-mode "a"는 추가모드 존재하는 파일에 내용을 추가할 수 있다.

os 모듈을 사용하여 Directory 다루기

```
import os
os.mkdir("log")
```

디렉토리가 있는지 확인하기

```
if not os.path.isdir("log"):
    os.mkdir("log")
```

Mac, window, linux 등의 운영체제는 Directory를 생성하는 명령이 다르기 때문에 OS 모듈을 사용한다.

Pickle

-파이썬의 객체를 영속화(계속 사용 할 수 있도록 객체화(고정)하는 것)하는 built-in 객체

```
import pickle

f = open("list.pickle", "wb")
test = [1, 2, 3, 4, 5]
pickle.dump(test, f)
f.close()

f = open("list.pickle", "rb")
test_pickle = pickle.load(f)
print(test_pickle)
f.close()
```

*test라는 list type 자체를 저장한다.