



## 예외 처리

오류가 발생했을 때, 회피하고 처리하는 기능

### 오류의 발생

오류는 어떤 상황에서 발생하는가?

- 0으로 나누는 경우
- 범위 밖 인덱스를 리칭하는 경우
- 없는 파일을 여는 경우 등등..

### 오류의 구성

오류 이름 : 오류의 내용으로 구성된다.

ex) `ZeroDivisionError` : division by zero

### 오류 예외 처리

#### try - except 문

↳ 코딩문과 비슷한 모양이다.

try :

실행할 문장 → 이제 코딩이라고 생각하라.

except [발생한 오류 as 오류변수] :

↓ 실행할 문장

오류가 발생하면 실행됨

ex) try :

4/0

except `ZeroDivisionError` as e

print(e)

# division by zero (오류 내용) 출력

## 여러개의 오류처리

try:

except 오류이름:

except 오류이름:

# 또는 except (오류1, 오류2) as e도 가능하다.

ex) try:

x = [1, 2]

print(x[3]) → 오류발생

4/0 → 이런 실행도 인덱스

except ZeroDivisionError as e:

print(e)

except IndexError as e:

print(e)

# list index out of range 안 출력

## finally와 else

try-except 구문에서 [ finally : try의 예외 발생 여부와 관계없이 실행  
else : try의 예외가 발생 안시킨 실행

ex) x = "a"

def my\_abs(x):

try:

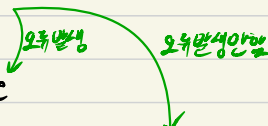
y = x > 1

except:

y = None

else:

y = (x > 0) \* x - (x < 0) \* x



## 2류 처리

pass 예약어로 오류를 회피할 수 있다.

```
ex) x = "1"
    try:
        x > 1
    except:
        pass
```

## 2류 발생시키기

직접 오류를 만들어낼 수도 있다.

### Exception 클래스 상속

```
class MyError (Exception):
    pass
```

ZeroDivisionError : Division by zero 처리

MyError : 이 생성되었다.

MyError를 발생시키려면 raise 예약어를 사용한다.

```
ex) def say_nick(nick):
    if nick == "바보":
        raise MyError()
```

```
    print(nick)
```

```
say_nick("천사")
```

```
say_nick("바보") → Error 발생
```

⚠ MyError 는 내용이 없는 상태다.

내용을 추가하려면?

--str-- 메소드 구현 필요

--str--

↳ MyError 의 내용을 출력할 때 실행되는 메소드

```
class MyError(Exception):
```

```
    def __str__(self):
```

```
        return "출력할 내용"
```

↳ print가 아닌 return 앞에 유의