파이썬 프로그래밍

예외 처리

한국기술교육대학교
온라인평생교육원

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- 예외가 발생할 수 있는 상황을 예상하여 예외 발생 상황을 전체 코드 흐름을 함께 제어할 수 있다.
- try/except/else/finally 절

- 구문

```
try:
    (예외 발생 가능한) 일반적인 수행문들
except Exception:
    예외가 발생하였을 때 수행되는 문들
else:
    예외가 발생하지 않았을 때 수행되는 문들
finally:
    예외 발생 유무와 관계없이 무조건 수행되는 문들
```

▪except 키워드 뒤 → 발생할 수 있는 exception 내용(클래스이름) 적음
▪수행문에서 Zerodivision 발생 예상 → exception에 Zerodivision 삽입
▪exception 밑에 존재하는 클래스에 해당하는 예외 전부 catch
▪try가 나오면 밑으로 finally 구문은 무조건 수행
▪else와 finally는 optional한 절

```
try:
    print 1.0 / 0.0
except ZeroDivisionError:
    print 'zero division error!!!'
```

```
zero division error!!!
```

▪try는 새로운 구문이 시작되어야 하기 때문에 콜론(:) 삽입
▪0으로 나누고 있기 때문에 ZeroDivisionError 발생
▪try, except 절 사용 X → 빨간 Error 발생 (프로그램 비정상적 수행)
▪try 절을 수행하면 프로그램이 정상적으로 수행되어 종료 X

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- 예외 처리를 하면 예외 발생시 프로그램 종료가 되지 않는다.

```
def division():
    for n in range(0, 5):
        try:
            print 10.0 / n
        except ZeroDivisionError, msg:
            print msg

division()
```

```
float division by zero
10.0
5.0
3.33333333333
2.5
```

▪msg 변수: ZeroDivisionError를 정의한 사람이 발생될 때 주는 메세지
▪예외 발생으로 프로그램이 종료되어 루프 진행 X
▪예외 처리를 하면 프로그램이 종료되지 않고 계속 진행됨

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- else: 구문은 except: 구문없이 사용 못한다.

```
def division():
    for n in range(0, 5):
        try:
            print 10.0 / n
        else:
            print "Success"

division()
```

```
  File "<ipython-input-49-e039cc968e23>", line 5
    else:
       ^
SyntaxError: invalid syntax
```

▪else는 except 없이는 사용 X

▪else가 있는데 except이 없는건 구문적 error → 빨간 X, 밑줄 제시

▪finally는 관계없이 적을 수 있음

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- 상황에 따라서는 에러와 함께 따라오는 정보를 함께 받을 수도 있다.

```
try:
    spam()
except NameError, msg:
    print 'Error -', msg
```

```
Error - name 'spam' is not defined
```

```
try:
    spam()
except NameError as msg:
    print 'Error -', msg
```

```
Error - name 'spam' is not defined
```

▪msg → NameError의 메시지가 구현됨
▪콤마(,) 대신 as 사용하여 메시지 받을 수 있음

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- try 절 안에서 간접적으로 호출한 함수의 내부 예외도 처리할 수 있다.

```
def zero_division():
    x = 1 / 0

try:
    zero_division()
except ZeroDivisionError, msg:
    print 'zero division error!!! -', msg
```

```
zero division error!!! -  integer division or modulo by zero
```

▪0으로 나누게 되니까 예외 발생
▪함수에는 예외 발생상황이 안보이지만 함수 수행 시 예외 발생 가능

```
def zero_division():
    x = 1 / 0

try:
    zero_division()
except ZeroDivisionError as msg:
    print 'zero division error!!! -', msg
```

```
zero division error!!! - integer division or modulo by zero
```

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- except 뒤에 아무런 예외도 기술하지 않으면 모든 예외에 대해 처리된다.

```
try:
    spam()
    print 1.0 / 0.0
except:
    print 'Error'
```

```
Error
```

▪NameError, NameEroor 둘 다 발생이 되는 상황
▪except 뒤에 아무런 내용 X → 모든 예외 catch
▪어떤 예외가 발생했는지는 알 수 없음

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

- 여러 예외들 각각에 대해 except 절을 다중으로 삽입할 수 있다.

```
b = 0.0
name = 'aaa.txt'
try:
    print 1.0 / b
    spam()
    f = open(name, 'r')
    '2' + 2
except NameError:
    print 'NameError !!!'
except ZeroDivisionError:
    print 'ZeroDivisionError !!!'
except (TypeError, IOError):
    print 'TypeError or IOError !!!'
else:
    print 'No Exception !!!'
finally:
    print 'Exit !!!'
```

```
ZeroDivisionError !!!
Exit !!!
```

▪ZeroDivisionError, NameError, IOError, TypeError 발생 가능

▪튜플 형태로 a or b로 묶어서 정의 가능

▪튜플은 가로가 없어도 사용 가능

▪각각의 경우에 따라서 제어를 해주고 싶으면 except 여러 번 사용 가능

▪예외가 발생하지 않아도 finally 이 부분은 항상 수행

▣ 예외 처리 방법

1. try/except/else/finally 절 사용하기

• 파일에서 숫자를 읽어와서 읽은 숫자로 나누기를 하는 예제
- 꼼꼼한 예외 처리 예제

```
import os
print os.getcwd()
filename = 't.txt'

try:
    f = open(filename, 'r')
except IOError, msg:
    print msg
else:
    a = float(f.readline())
    try:
        answer = 1.0 / a
    except ZeroDivisionError, msg:
        print msg
    else:
        print answer
    finally:
        print "Finally!!!"
        f.close()
```

```
/Users/yhhan/git/python-e-learning
1.0
Finally!!!
```

▪os.getcwd() → 현재 디렉토리를 알려줌
▪f = open이라는 구문은 try, except 사이에 항상 들어와야 함
▪file 이름을 잘못 적거나 파일이 없을 수도 있으므로 미리 방지 가능
▪보통 파일 안에 어떤 것이 있는지 모르므로 방지를 위해 사용

▣ 예외 처리 방법

2. 같은 부류의 예외 다 잡아내기

- 예외 클래스들은 상속에 의한 계층 관계를 지니고 있기 때문에 이를 이용하면 여러 예외들을 한꺼번에 잡을 수 있다.
- 예를 들어, ArithmeticError의 하위 클래스로서 FloatingPointError, OverflowError, ZeroDivisionError가 존재하기 때문에 이들 하위 클래스 예외가 발생하였을 경우 ArithmeticError로서 잡아낼 수 있다.

```
def dosomething():
    a = 1/0

try:
    dosomething()
except ArithmeticError:
    print "ArithmeticException occured"
```

```
ArithmeticException occured
```

▪ArithmeticError의 하위클래스로 같은 부류

▪except 옆에 상위 클래스 적으면 하위 클래스의 Error 모두 catch

▣ 예외 처리 방법

2. 같은 부류의 예외 다 잡아내기

- 예외가 임의의 except에 의해 잡히면 다른 except에 의해서는 잡히지 않는다.

```
def dosomething():
    a = 1/0

try:
    dosomething()
except ZeroDivisionError:  # ZeroDivisionError는 이곳에서 잡힌다.
    print "ZeroDivisionError occured"
except ArithmeticError:    # FloatingPointError, OverflowError는 이곳에서 잡힌다.
    print "ArithmeticException occured"
```

```
ZeroDivisionError occured
```

▪하위 클래스 예외를 정의하면, 상위 클래스 예외 정의 시 같이 적용 X

▣ 예외 처리 방법

2. 같은 부류의 예외 다 잡아내기

```
def dosomething():
    a = 1/0

try:
    dosomething()
except ArithmeticError:
    print "ArithmeticException occured"
except ZeroDivisionError:  # 이곳에서 ZeroDivisionError는 잡히지 않는다.
                                ==> 잘못된 코드
    print "ZeroDivisionError occured"
```

```
ArithmeticException occured
```

▪하위 클래스가 상위 클래스 안에 있는 것으로 상위클래스만 수행됨
▪아래 따로 정의한 하위 클래스 내용은 절대 수행 X