Python Programming

7. 예외 처리

Tacademy

CONTENTS

1 예외 처리란

2 예외 처리의 종류

3 사용자 정의 예외 및 기타 구문

학습 목표

- 파이썬의 예외 처리에 대해 이해할 수 있습니다.
- 여러 가지 예외의 종류에 대해서 이해할 수 있습니다.
- 사용자 정의 예외와 raise, assert의 사용법에 대해서 이해합니다.

예외(Exception)란?

프로그램의 제어 흐름을 조정하기 위해 사용하는 이벤트

- 처리를 하지 않는 예외에 대하여 자동으로 에러(Error)가 발생하고 프로그램을 종료

```
>>> a = [10, 20, 30]
>>> a[3]   # 리스트의 크기를 넘어서는 인덱스 참조
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a[3]   # 리스트의 크기를 넘어서는 인덱스 참조
IndexError: list index out of range
```

처리되지 않은 예외(Unhandled Exception)

- '0'으로 나누는 경우
- 원격에 있는 데이터 베이스 접속시 연결되지 않는 경우
- 파일을 열었는데 사용자에 의해서 삭제된 경우

구문 에러

- 오타, 들여쓰기의 실수로 발생
- 인터프리터에서 에러가 의심되는 부분을 개발자에게 알려줌

```
>>> print("{0}, {1}.format(10, 20))
  File "<stdin>", line 1
    print("{0}, {1}.format(10, 20))
                                  ^
SyntaxError: EOL while scanning string literal
>>> if a > 5 print('a is bigger than 5')
  File "<stdin>", line 1
    if a > 5 print('a is bigger than 5')
             ^
SyntaxError: invalid syntax
>>>
```

예외의 몇 가지 종류(1)

NameError 선언하지 않은 변수 'a'에 접근

```
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
```

ZeroDivisionError '0'으로 나눔

```
>>> 10 / 0
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    10 / 0
ZeroDivisionError: int division or modulo by zero
```

예외의 몇 가지 종류(2)

IndexError 리스트의 접근 가능한 인덱스를 넘음

```
>>> a = [10, 20, 30]
>>> a[3]
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    a[3]
IndexError: list index out of range
```

TypeError 지원하지 않는 연산(정수를 문자열로 나눔)

```
>>> a = 'Apple'
>>> 10 / a
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    10 / a
TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

내장 예외 클래스 계층구조

내장 예외는 exceptions 모듈에 미리 정의

- 프로그램 동작 중 자동적으로 발생
- 개발자가 명시적으로 예외 발생도 가능

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EnvironmentError
      |    +-- IOError
      |    +-- OSError
      |         +-- WindowsError (Windows)
      |         +-- VMSError (VMS)
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

주요 내장 예외

클래스 이름	내용
Exception	모든 내장 예외의 기본 클래스 - 사용자 정의 예외를 작성시 활용
ArithmeticError	수치 연산 예외의 기본 클래스
LookupError	시퀀스 관련 예외의 기본 클래스
EnvironmentError	파이썬 외부 에러의 기본 클래스

참고 자료

- 빠르게 활용하는 파이썬 3, 예외 처리 챕터
- 파이썬 공식 문서, Built-in Exceptions 부분

예외 처리(1)

try 구문

```
try:
  <예외 발생가능성이 있는 문장>
except <예외 종류> :
  <예외 처리 문장>
except (예외 1, 예외 2):
  <예외 처리 문장>
except 예외 as 인자:
  <예외 처리 문장>
else:
  <예외가 발생하지 않은 경우, 수행할 문장>
finally:
  <예외 발생 유무에 상관없이
    try 블록 이후 수행할 문장>
```

예외 처리(2)

try ~ except 예제

```
def divide(a, b):
    return a / b

try:
    c = divide(5, 0)
except:
    print('Exception is occured!!')
```

수행 결과

```
Exception is occured!!
```

예외 처리(3)

다양한 예외 처리

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 'string')
except ZeroDivisionError:
    print('두번째 인자는 0이면 안됩니다.')
except TypeError:
    print('모든 인수는 숫자이어야 합니다.')
except:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

수행 결과

```
>>>
모든 인수는 숫자이어야 합니다.
```

예외 처리(4)

Else와 finally 예제

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 2)
except ZeroDivisionError:
    print('두번째 인자는 0이면 안됩니다.')
except TypeError:
    print('모든 인수는 숫자이여야 합니다.')
except:
    print('ZeroDivisionError, TypeError를 제외한 다른 에러')
else:# 예외가 발생하지 않는 경우
    print('Result: {0}'.format(c))
finally: # 예외 발생 유무와 상관없이 수행
    print('항상 finally 블록은 수행됩니다.')
```

수행 결과

```
Result: 2.5
항상 finally 블록은 수행됩니다.
```

예외 처리(5)

예외에 대한 정보를 전달받는 예제

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, "af")
except TypeError as e: # 전달되는 예외 인스턴스 객체를 e로 받아서 사용
    print('에러: ', e.args[0])
except Exception:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

수행 결과

```
에러:  unsupported operand type(s) for /: 'int' and 'str'
```

예외 처리(6)

예외를 묶어서 처리하는 예제

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 0)
except (ZeroDivisionError, OverflowError, FloatingPointError):
    # 명시된 에러를 모두 처리
    print('수치 연산 관련 에러입니다.')
except TypeError:
    print('모든 인수는 숫자이어야 합니다.')
except Exception:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

수행 결과

```
수치 연산 관련 에러입니다.
```

예외 처리(7)

상위 예외 클래스를 처리하는 예제

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 0)
except ArithmeticError:
    #상위 클래스를 처리시 하위 모든
    # 클래스도 이 부분에서 처리
    print('수치 연산 관련 에러입니다.')
except TypeError:
    print('모든 인수는 숫자이어야 합니다.')
except Exception:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

예외 처리(8)

try ~ finally

try:
 <예외 발생 가능성이 있는 문장>
finally:
 <예외와 관계없이, 항상 수행되어야 할 문장>

예제

```
# -*- coding: cp949 -*-
FilePath = './test.txt'

try:
    f = open(FilePath, 'r')
    try:
        data = f.read(128)
        print(data)
    finally:
        f.close()
except IOError:
    print("Fail to open {0} file".format(FilePath))
```

raise 구문

◉ 명시적으로 예외 발생

raise 구문 형식

- raise [Exception]
- raise [Exception(data)]
- raise

예제

```
# -*- coding: cp949 -*-
def RaiseErrorFunc():
    raise NameError # 내장 예외인 NameError를 발생

try:
    RaiseErrorFunc()
except:
    print("NameError is Catched")
```

사용자 정의 예외(1)

내장 예외만으로 부족한 경우,
개발자가 직접 예외를 정의하여 사용 가능

- Exception 클래스나 그 하위 클래스를 상속받아서 구현
- 생성자에 클래스 멤버 변수를 이용하여
  인자를 에러 처리부로 전달

예제

```
class NegativeDivisionError(Exception): # 사용자 정의 예외 정의
    def __init__(self, value):
        self.value = value
```

사용자 정의 예외(2)

예제

```
def PositiveDivide(a, b):
    if(b < 0):
        raise NegativeDivisionError(b)
    return a / b

try:
    ret = PositiveDivide(10, -3)
    print('10 / 3 = {0}'.format(ret))
except NegativeDivisionError as e: # 사용자 정의 예외인 경우
    print('Error - Second argument of PositiveDivide is ', e.value)
except ZeroDivisionError as e: # '0'으로 나누는 경우
    print('Error - ', e.args[0])
except: # 그 외 모든 예외의 경우
    print(e.args)
```

수행 결과

```
Error - Second argument of PositiveDivide is  -3
```

assert 구문(1)

표현식	Assert 〈조건식〉, 〈관련 데이터〉

인자로 받은 조건식이 거짓인 경우, AssertionError가 발생

- 개발과정에서 디버깅, 제약 사항 설정 등으로 사용
- __debug__가 True인 경우만 assert 구문 활성화
  - 명령 프롬프트에서 최적화 옵션(-O)을 설정하면 __debug__는 False로 설정됨
  - 다음 코드와 동일

```
if __debug__:
    if not <조건식>:
            raise AssertionError(<관련 데이터>)
```

assert 구문(2)

예외에 대한 정보를 전달받는 예제

```
# -*- coding: cp949 -*-
def foo(x): # 받은 인자의 type이 정수형인지 검사
    assert type(x) == int, "Input value must be integer"
    return x * 10

ret = foo("a") # AssertionError가 발생
print(ret)
```

수행 결과

```
C:\Python31>python.exe 7-6-1.py
Traceback (most recent call last):
  File "7-6-1.py", line 6, in <module>
    ret = foo("a") # AssertionError가 발생
  File "7-6-1.py", line 3, in foo
    assert type(x) == int, "Input value must be integer"
AssertionError: Input value must be integer

C:\Python31>python.exe -O 7-6-1.py
aaaaaaaaaa
```

실용 예제 (1)

Tree와 거의 동일하게 작성 된 Python Code

tree.py

```
1  import glob, os.path
2
3  def traverse(dir, depth):
4      for obj in glob.glob(dir+'/*'):
5          if depth==0:
6              prefix = '|--'
7          else:
8              prefix = '|' + ' '*depth + '+--'
9          if os.path.isdir(obj):
10             print( prefix + os.path.basename(obj) )
11             traverse( obj, depth+1 )
12         elif os.path.isfile(obj):
13             print( prefix + os.path.basename(obj) )
14         else:
15             print( prefix + 'unknown object:', obj)
16
17 if __name__=='__main__':
18     traverse('.', 0)
19
```

=

12장 예제 미리보기

해당 디렉토리의 구조를 화면에 출력하는 기능.

실용 예제 (2)

Raise를 활용한 응답방법

```
def traverse(dir, depth):
        for obj in glob.glob(dir+'/*'):
                if depth==0:
                        prefix = '|--'
                else:
                        prefix = '|' + ' '*depth + '+--'
                if os.path.isdir(obj):
                        print( prefix + os.path.basename(obj) )
                        traverse( obj, depth+1 )
                elif os.path.isfile(obj):
                        print( prefix + os.path.basename(obj) )
                else:
                        #print( prefix + 'unknown object:', obj)
                        raise UnknownObjectError(obj)
```

실용 예제 (3)

Exc_info 활용

```
if __name__=='__main__':
        try:
                traverse('.', 0)
        except UnknownObjectError as e:
                print('UnknownObjectError occurs:', e.obj)
        except:
                exc, value, tb = sys.exc_info()
                print(exc, value, tb)
                traceback.print_exc()
```

실용 예제 (4)

Traceback 객체 사용

```
C:₩Users₩dsp₩Dropbox₩python3.5초급>python tree_exception.py
<class 'NameError'> name 'p' is not defined <traceback object at 0x007A6058>
Traceback (most recent call last):
  File "tree_exception.py", line 26, in <module>
    p
NameError: name 'p' is not defined
```

학습정리

지금까지 [예외 처리]에 대해서 살펴보았습니다.

예외란

- 예외의 정의
- 예외의 종류들 및 예외 클래스 구조

예외처리 구문 사용 방법

- Try, except, else, finally 구문 사용법

사용자 정의 예외 및 기타 구문

- Raise, assert 구문 사용법
- 사용자 정의 예외처리 방법