학습 내용

2부. 프로그래밍 언어 활용

6장. 모듈과 패키지

7장. 객체지향 프로그래밍

8장. 예외 처리

- 1. 예외 처리
- 2. try~except 구문으로 예외 처리하기
- 3. raise로 예외 발생시키기
- 4. raise로 추상클래스 정의하기
- 5. 사용자 정의 예외
- 6. 정리 작업

9장. 파일 입/출력 프로그래밍

10장. 데이터베이스 연동

예외 처리

1절. 예외처리

에러	문법에러	문법적으로 나타나는 에러
	실행에러 (실행 시 에러)	논리에러 : 프로그래머가 잘못 작성해서 이상한 결과가 나오는 경우
		시스템에러 : 프로그래머의 의지와 상관없이 나타나는 에러
		예외사항 : 정상적으로 프로그램이 동작하는 과정에서 나타나는 에러 프로그램 실행 중에 발생하는 예기치 않는 사건 ex. 파일을 다룰 때 팔일이 없거나 쓰기 금지로 인한 오류 ex. 네트워크 프로그래밍 시 네트워크 연결 실패로 인한 오류 ex. 리스트 또는 튜플의 인덱스를 벗어난 경우 ex. 정수를 0으로 나누는 경우 ex. 부적절한 형변환이 일어나는 경우 ex. 입출력을 위한 파일이 없는 경우

예외 처리

- 예외 처리(Exception Handling) : 문제가 없을 것 같은 프로그램도 외부 환경 요인등에 의해서 문제가 발생하곤 합니다. 프로그램에서 문제가 발생될 만한 곳을 예상하여 사전에 "문제가 발생하면 이렇게 하라" 라고 미리 프로그래밍 하는 것

예외의 전형적인 예

1절. 예외처리

```
1  f = open('없는파일.txt', 'r')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-c0c01e0cc22c> in <module>()
----> 1 f = open('없는파일.txt', 'r')

FileNotFoundError: [Errno 2] No such file or directory: '없는파일.txt'
```

파일을 다룰 때 파일이 없거나 쓰기 금지로 인한 오류

```
1  4 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-87ac9c94a8c2> in <module>()
----> 1 4 / 0

ZeroDivisionError: division by zero
```

0으로 나누려는 오류

```
1  a = [1,2,3]
2  a[3]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-3-47894dcd86df> in <module>()
      1 a = [1,2,3]
----> 2 a[3]

IndexError: list index out of range
```

리스트의 인덱스를 벗어난 참조 오류

try ~ except

2절. try~except 구문으로 예외 처리하기

- try는 예외를 처리할 때 예외가 발생할 가능성이 있는 문장을 작성
  - try 절(또는 블록)에서 예외가 발생하면 except 절을 찾음
- except 절은 예외가 발생했을 경우 실행되어야 할 코드를 작성

```
try :
    # 예외가 발생할 가능성이 있는 문장
except :
    # 예외가 발생했을 경우 실행할 문장
```

try ~ except

2절. try~except 구문으로 예외 처리하기

- 유효한 정수가 입력되면 try 블록의 나머지 문장을 실행
- 입력한 값이 정수가 아니면 정수형으로 변환하는 도중 예외가 발생
  - try 블록에서 예외가 발생하면 해당 예외를 처리할 수 있는 except 블록으로 제어가 이동

```
while True:
    try:
        x = int(input('정수를 입력하세요: '))
        print('입력한 정수는 {}입니다.'.format(x))
        break
    except:
        print('유효한 정수가 아닙니다. 다시 시도하세요.')
```

```
정수를 입력하세요: hello
유효한 정수가 아닙니다. 다시 시도하세요.
정수를 입력하세요: 12345
입력한 정수는 12345입니다.
```

예외를 지정한 처리

2절. try~except 구문으로 예외 처리하기

- except 절은 예외 이름을 생략 할 수 있음
  - 프로그래밍 오류를 쉽게 처리 할 수 있지만 실제로 어떤 오류를 처리하는지는 알 수 없음
  - 가능하다면 except 절에 예외를 지정해서 사용하는 것이 좋음

```
try :
    # 예외가 발생할 가능성이 있는 문장
except Exception :
    # 예외가 발생했을 경우 실행할 문장
```

예외를 지정한 처리

2절. try~except 구문으로 예외 처리하기

- try 절의 실행 중에 예외가 발생하면 예외가 발생한 이후의 문장들이 실행되지 않음
- 발생한 예외의 유형이 except 키워드 다음에 명명 된 예외와 일치하면 해당 except 절이 실행
- 예외를 처리할 except 절이 발견 되지 않으면 처리되지 않은 예외 이며 프로그램 실행은 중지됨

```
while True:
    try:
        x = int(input('정수를 입력하세요: '))
        print('입력한 정수는 {}입니다.'.format(x))
        break
    except ValueError:
        print('유효한 정수가 아닙니다. 다시 시도하세요.')
```

예외 별로 처리하기

- try 블록에서 발생할 수 있는 예외가 여러 개일 경우 각각의 예외를 처리하도록 catch 블록을 정의해 놓을 수 있음

```
try :
    # 예외가 발생할 가능성이 있는 문장
except Error1 :
    # Error1이 발생했을 경우 실행할 문장
except Error2 :
    # Error2가 발생했을 경우 실행할 문장
```

예외 별로 처리하기

```
while True:
    try:
        x = int(input('정수를 입력하세요: '))
        print('입력한 정수는 {}입니다.'.format(x))
        print('100을 입력한 수로 나누면 {}입니다.'.format(100/x))
        break
    except ValueError:
        print('유효한 정수가 아닙니다. 다시 시도하세요.')
    except ZeroDivisionError:
        print('0으로 나눌 수 없습니다.')
```

```
정수를 입력하세요: 0
입력한 정수는 0입니다.
0으로 나눌 수 없습니다.
정수를 입력하세요: hello
유효한 정수가 아닙니다. 다시 시도하세요.
정수를 입력하세요: 20
입력한 정수는 20입니다.
100을 입력한 수로 나누면 5.0입니다.
```

- except 구문의 에러(또는 예외) 클래스는 예외클래스 상속관계에서 부모의 예외가 나중에 나와야 함
- 만일 상위의 예외를 먼저 사용하면 이후의 catch 블록은 실행되지 않는 블록이 됨

다중 예외 처리하기

2절. try~except 구문으로 예외 처리하기

- except 절에 여러 예외를 지정
- except 절은 여러 예외를 괄호로 묶은 튜플로 지정 할 수 있음

```
except (Error1, Error2, ...):
    pass
```

```
while True:
    try:
        x = int(input('정수를 입력하세요: '))
        print('입력한 정수는 {}입니다.'.format(x))
        print('100을 입력한 수로 나누면 {}입니다.'.format(100/x))
        break
    except (ValueError, ZeroDivisionError):
        print('유효한 수가 아닙니다. 다시 시도하세요.')
```

else

- else절은 try절이 예외를 발생시키지 않을 때 실행해야하는 코드에 사용
- else 절은 except 절 다음에 와야 함

```
try:
    f = open('myfile.txt', 'r')
except FileNotFoundError:
    print('파일이 없습니다.')
else:
    data = f.read()
    print(data)
```

myfile.txt 파일이 없을 경우

```
파일이 없습니다.
```

myfile.txt 파일의 내용이 hello 일 때

```
hello myfile
```

else 절을 사용한 코드

```
try:
    f = open('myfile.txt', 'r')
    data = f.read()
    print(data)
except FileNotFoundError:
    print('파일이 없습니다.')
```

myfile.txt 파일이 없을 경우

```
파일이 없습니다.
```

myfile.txt 파일의 내용이 hello 일 때

```
hello myfile
```

else 절을 사용하지 않은 코드

예외 인수

2절. try~except 구문으로 예외 처리하기

- 예외가 발생하면 관련 값이 있음
- 이를 예외의 인수라고 부름
- 예외 객체 변수에는 *instance*.args를 가지고 있어 예외 관련 정보들을 조회할 수 있음

```
while True:
    try:
        x = int(input('정수를 입력하세요: '))
        print('입력한 정수는 {}입니다.'.format(x))
        print('100을 입력한 수로 나누면 {}입니다.'.format(100/x))
        break
    except (ValueError, ZeroDivisionError) as e:
        print("예외유형", type(e))
        print("예외 메시지", e.args)
        print("예외", e)
```

```
정수를 입력하세요: 0
입력한 정수는 0입니다.
예외유형 <class 'ZeroDivisionError'>
예외 메시지 ('division by zero',)
예외 division by zero
정수를 입력하세요: hello
예외유형 <class 'ValueError'>
예외 메시지 ("invalid literal for int() with base 10: 'hello'",)
예외 invalid literal for int() with base 10: 'hello'
정수를 입력하세요: 20
입력한 정수는 20입니다.
100을 입력한 수로 나누면 5.0입니다.
```

raise

3절. raise로 예외 발생시키기

- raise는 강제로 예외를 발생해야 할 필요가 있을 때 사용
- raise 명령문을 사용하면 프로그래머는 지정된 예외가 발생하도록 할 수 있음

```
raise NameError('예외가 발생했어요.')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-b6b2c2c1f827> in <module>
----> 1 raise NameError('예외가 발생했어요.')

NameError: 예외가 발생했어요.
```

예외처리 메커니즘

3절. raise로 예외 발생시키기

호출자

예외의 원인을 제공 했으므로
이곳에서 예외를 처리함

```
data = []
try:
    insert(data)
except Exception as e:
    print(e.args[0])
else:
    print("정상 실행되었습니다.")
```

피 호출자

data의 길이가 0이면
예외를 발생시킴

```
def insert(data):
    if len(data)==0:
        raise Exception("길이가 0입니다.")
    print(data, "을(를) 입력합니다.", sep=" ")
```

raise와 try~except

```
# 피호출자(callee)
def insert(data):
    if len(data)==0:
        raise Exception("길이가 0입니다.")
    print(data, "을(를) 입력합니다.", sep=" ")
```

```
# 호출자(caller)
# data = [1, 2, 3]
data = []
try:
    insert(data)
except Exception as e:
    print(e.args[0])
else:
    print("정상 실행되었습니다.")
```

길이가 0입니다.

사용자 정의 예외를 이용한 예외처리

5절. 사용자 정의 예외

```
class ZeroLengthError(Exception):
    """길이가 0일 때 발생하는 예외."""
    def __init__(self):
        Exception.__init__(self, "사용자 정의 예외 : 길이가 0입니다.")
```

사용자 정의 예외 클래스

```
def insert(data):
    if len(data)==0:
        raise ZeroLengthError()
    print(data, "을(를) 입력합니다.", sep=" ")
```

예외를 발생시킴

```
# data = [1, 2, 3]
data = []
try:
    insert(data)
except Exception as e:
    print(e.args[0])
else:
    print("정상 실행되었습니다.")
```

예외를 처리함

```
사용자 정의 예외 : 길이가 0입니다.
```

6절. 정리 작업

- 자원을 반납하는 코드를 작성하여 프로그램을 더 안정적으로 코딩
- 파일 정리작업 방법
  1) 예외처리 시 사용하는 finally 블록을 이용
  2) 사전 정의된 정리작업을 위한 with를 이용

```
try:
    f = open('myfile.txt', 'r')
    #1 파일이 열린 후 실행되어야 하는 코드
except FileNotFoundError as e:
    print(str(e))
else:
    #2 파일을 읽기 전에 실행해야 하는 코드
    data = f.read()
```

finally 이용

```
try:
    f = open('myfile.txt', 'r')
except FileNotFoundError as e:
    print(str(e))
else:
    data = f.read()
    print(data)
finally:
    f.close()
```

```
hello myfile
```

Finally 블록은 예외 발생
유/무와상관없이 항상 실행

with 이용

6절. 정리 작업

```
for line in open("myfile.txt"):
    print(line, end="")
```

print 구문을 실행한 후 불확실한 시간 동안 파일이 열려 있음

```
hello myfile
```

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

파일 f가 항상 닫힘

```
hello myfile
```

7절 연습문제(실습형)

1. 숫자 두 개를 입력받아 나눗셈 연산을 하는 프로그램을 작성하세요.
   - 두 숫자는 정수 또는 실수일 수 있으며, 0으로 나눌 수 없습니다.
   - 올바른 두 수를 입력하고 나눗셈 연산 결과를 출력한 후 종료해야합니다.
   - 다음은 실행 후 출력 예시입니다.

```
첫 번째 숫자를 입력하세요: hello
유효한 숫자가 아닙니다. 다시 시도하세요.
첫 번째 숫자를 입력하세요:  10
두 번째 숫자를 입력하세요: 0
입력한  수는 10.0와 0.0입니다.
유효한 숫자가 아닙니다. 다시 시도하세요.
첫 번째 숫자를 입력하세요:  3.5
두 번째 숫자를 입력하세요: hello
유효한 숫자가 아닙니다. 다시 시도하세요.
첫 번째 숫자를 입력하세요:  3.5
두 번째 숫자를 입력하세요: 2
입력한 수는 3.5와 2.0입니다.
3.5을 2.0로 나누면  1.75입니다.
```

7절 연습문제(문제풀이형)

1. 다음 중 예외처리에 대해 잘 못 설명한 것은?

① try 블록은 예외가 발생할 가능성이 있는 문장을 작성한다.

② 예외가 발생하면 except 블록이 실행된다.

③ 상위 예외처리를 위한 except 블록은 하위 예외처리 except 블록에 비해 먼저 선언되 어야 한다.

④ finally 블록은 예외의 발생 유/무와 상관없이 실행된다.

2. 다음 중 예외처리에 대한 설명 중 잘못된 것은?

① raise는 강제로 예외를 발생시킬 때 사용한다.

② catch 절은 예외를 처리하기 위해 사용하는 구문이다.

③ 다른 예외를 하나의 예외처리 구문으로 처리할 수 있다.

④ else 절은 예외가 발생하지 않을 경우 실행된다.

7절 연습문제(문제풀이형)

3. 다음 중 except 절을 잘못 사용한 것은?

① except:

② except Exception:

③ except Exception as e:

④ except Exception e:

4. 다음 중에서 예외처리에 사용하지 않는 구문은?

① try

② except

③ with

④ finally

```
+--Exception
     +--StopIteration
     +--StopAsyncIteration
     +--ArithmeticError
     ¦    +--FloatingPointError
     ¦    +--OverflowError
     ¦    +--ZeroDivisionError
     +--AssertionError
     +--AttributeError
     +--BufferError
     +--EOFError
     +--ImportError
          +--ModuleNotFoundError
     +--LookupError
     ¦    +--IndexError
     ¦    +--KeyError
     +--MemoryError
     +--NameError
     ¦    +--UnboundLocalError
     +--OSError
     ¦    +--BlockingIOError
     ¦    +--ChildProcessError
     ¦    +--ConnectionError
     ¦    ¦    +--BrokenPipeError
     ¦    ¦    +--ConnectionAbortedError
```

```
+--Exception
     ¦    ¦    +--ConnectionRefusedError
     ¦    ¦    +--ConnectionResetError
     ¦    +--FileExistsError
     ¦    +--FileNotFoundError
     ¦    +--InterruptedError
     ¦    +--IsADirectoryError
     ¦    +--NotADirectoryError
     ¦    +--PermissionError
     ¦    +--ProcessLookupError
     ¦    +--TimeoutError
     +--ReferenceError
     +--RuntimeError
     ¦    +--NotImplementedError
     ¦    +--RecursionError
     +--SyntaxError
     ¦    +--IndentationError
     ¦         +--TabError
     +--SystemError
     +--TypeError
     +--ValueError
     ¦    +--UnicodeError
     ¦         +--UnicodeDecodeError
     ¦         +--UnicodeEncodeError
     ¦         +--UnicodeTranslateError
     +--Warning
          +--DeprecationWarning
          +--PendingDeprecationWarning
          +--RuntimeWarning
          +--SyntaxWarning
          +--UserWarning
          +--FutureWarning
          +--ImportWarning
          +--UnicodeWarning
          +--BytesWarning
          +--ResourceWarning
```