08. 예외 처리

- 예외란
- try ~ except로 예외 처리하기
  - 복수 개의 except절 사용하기
  - try절을 무사히 실행하면 만날 수 있는 else
  - 어떤 일이 있어도 반드시 실행되는 finally
- Exception 클래스
- 우리도 예외 좀 일으켜보자
- 내가 만든 예외 형식

예외란

- 파이썬에서 예외(Exception)는 문법적으로는 문제가 없는 코드를 실행하는 중에 발생하는 오류
- 실습 1 (ValueError 일으키기)

```
>>> def my_power(y):
        print("숫자를 입력하세요.")
        x = input()
        return int(x) ** y
>>> my_power(2)
숫자를 입력하세요.
3
9
>>> my_power(3)
숫자를 입력하세요.
abc
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    my_power(3)
  File "<pyshell#18>", line 4, in my_power
    return int(x) ** y
ValueError: invalid literal for int() with base 10: 'abc'
```

try ~ except로 예외 처리하기

- try 절 안에 오류가 발생 할 수 있는 코드 블록을 배치하고
- except 절에는 문제가 생겼을 때 뒤처리를 하는 코드 블록 배치

```
try:
  # 문제가 없을 경우 실행할 코드 (예외가 발생 할 수 있는 코드)
except:
  # 문제가 생겼을 때 실행할 코드
```

실습 1

```
>>> 1/0
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    1/0
ZeroDivisionError: division by zero
```

예제 : 10/try_except.py

```
try:
    print(1/0)

except:
    print("예외가 발생했습니다.")
```

- 실행 결과

```
>try_except.py
예외가 발생했습니다.
```

try ~ except로 예외 처리하기 – 복수 개의 except절 사용하기

❖ try 블록 안에서 여러 종류의 예외가 발생하는 경우에 사용

```
try:
  # 문제가 없을 경우 실행할 코드
except 예외형식1(발생 오류):
# 문제가 생겼을 때 실행할 코드
except 예외형식2(발생 오류):
# 문제가 생겼을 때 실행할 코드
```

❖ 예제 : 10/multiple_except.py

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

try:
    print("첨자를 입력하세요:")
    index = int(input())
    print(my_list[index]/0)
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("잘못된 첨자입니다.")
```

index가 0~2사이로 입력된 다면 ZeroDivisionError가 일어납니다.

index가 0~2를 벗어나면 my_list[index]에서 IndexError가 발생합니다.

- 실행 결과

```
>multiple_except.py
첨자를 입력하세요:
2
0으로 나눌 수 없습니다.

>multiple_except.py
첨자를 입력하세요:
10
잘못된 첨자입니다.
```

try ~ except로 예외 처리하기 – 복수 개의 except절 사용하기

❖ 예외의 인스턴스를 활용하는 방법 : as 문 사용

-예외 사항에 대한 세부 정보 얻고 싶을 때 (이름에서 예외 객체 전체를 얻음)

```
try:
  # 문제가 없을 경우 실행할 코드
except 예외형식1 (발생 오류)  as err (오류 변수):
# 문제가 생겼을 때 실행할 코드
except 예외형식2 as err:
# 문제가 생겼을 때 실행할 코드
```

❖ 예제 : 10/multiple_except2.py

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

try:
    print("첨자를 입력하세요:")
    index = int(input())
    print(my_list[index]/0)
except ZeroDivisionError as err:
    print("0으로 나눌 수 없습니다. ({0})".format(err))
except IndexError as err:
    print("잘못된 첨자입니다. ({0})".format(err))
```

- 실행 결과

```
>multiple_except2.py
첨자를 입력하세요:
2
0으로 나눌 수 없습니다.
(division by zero)

>multiple_except2.py
첨자를 입력하세요:
10
잘못된 첨자입니다. (list
index out of range)
```

try ~ except로 예외 처리하기 – try절을 무사히 실행하면 만날 수 있는 else

❖ try에 대한 else가 아닌 "except절에 대한 else"

```
try:
    # 실행할 코드블록
except:
    # 예외 처리 코드블록
else:
    # except절을 만나지 않았을 경우 실행하는 코드블록
```

❖ 예제 : 10/try_except_else.py

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

try:
    print("첨자를 입력하세요:")
    index = int(input())
    print("my_list[{0}]: {1}".format(index, my_list[index]))
except Exception as err:
    print("예외가 발생했습니다 ({0})".format(err))
else:
    print("리스트의 요소 출력에 성공했습니다.")
```

- 실행 결과

```
>try_except_else.py
첨자를 입력하세요:
1
my_list[1]: 2
리스트의 요소 출력에 성공했습니다.

>try_except_else.py
첨자를 입력하세요:
10
예외가 발생했습니다 (list index out of range)
```

try ~ except로 예외 처리하기 – 어떤 일이 있어도 반드시 실행되는 finally

- **finally는 예외가 발생했든 아무 일이 없든 간에 "무조건" 실행**
  - finally는 파일이나 통신 채널과 같은 컴퓨터 자원을 정리할 때 요긴하게 사용됨.

- **예제 : 10/try_except_finally.py**

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

try:
    print("첨자를 입력하세요:")
    index = int(input())
    print("my_list[{0}]: {1}".format(index, my_list[index]))
except Exception as err:
    print("예외가 발생했습니다 ({0})".format(err))
finally:
    print("어떤 일이 있어도 마무리합니다.")
```

- 실행 결과

```
>try_except_finally.py
첨자를 입력하세요:
2
my_list[2]: 3
어떤 일이 있어도 마무리합니다.

>try_except_finally.py
첨자를 입력하세요:
10
예외가 발생했습니다 (list index out of range)
어떤 일이 있어도 마무리합니다.
```

try ~ except로 예외 처리하기 – 어떤 일이 있어도 반드시 실행되는 finally

❖ **finally가 else는 함께 사용하는 것도 가능함.**

```
try:
    # 코드 블록
except:
    # 코드블록
else:
    # 코드블록
finally:
    # 코드블록
```

❖ **예제 : 10/try_except_else_finally.py**

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

try:
    print("첨자를 입력하세요:")
    index = int(input())
    print("my_list[{0}]: {1}".format(index, my_list[index]))
except Exception as err:
    print("예외가 발생했습니다 ({0})".format(err))
else:
    print("리스트의 요소 출력에 성공했습니다.")
finally:
    print("어떤 일이 있어도 마무리합니다.")
```

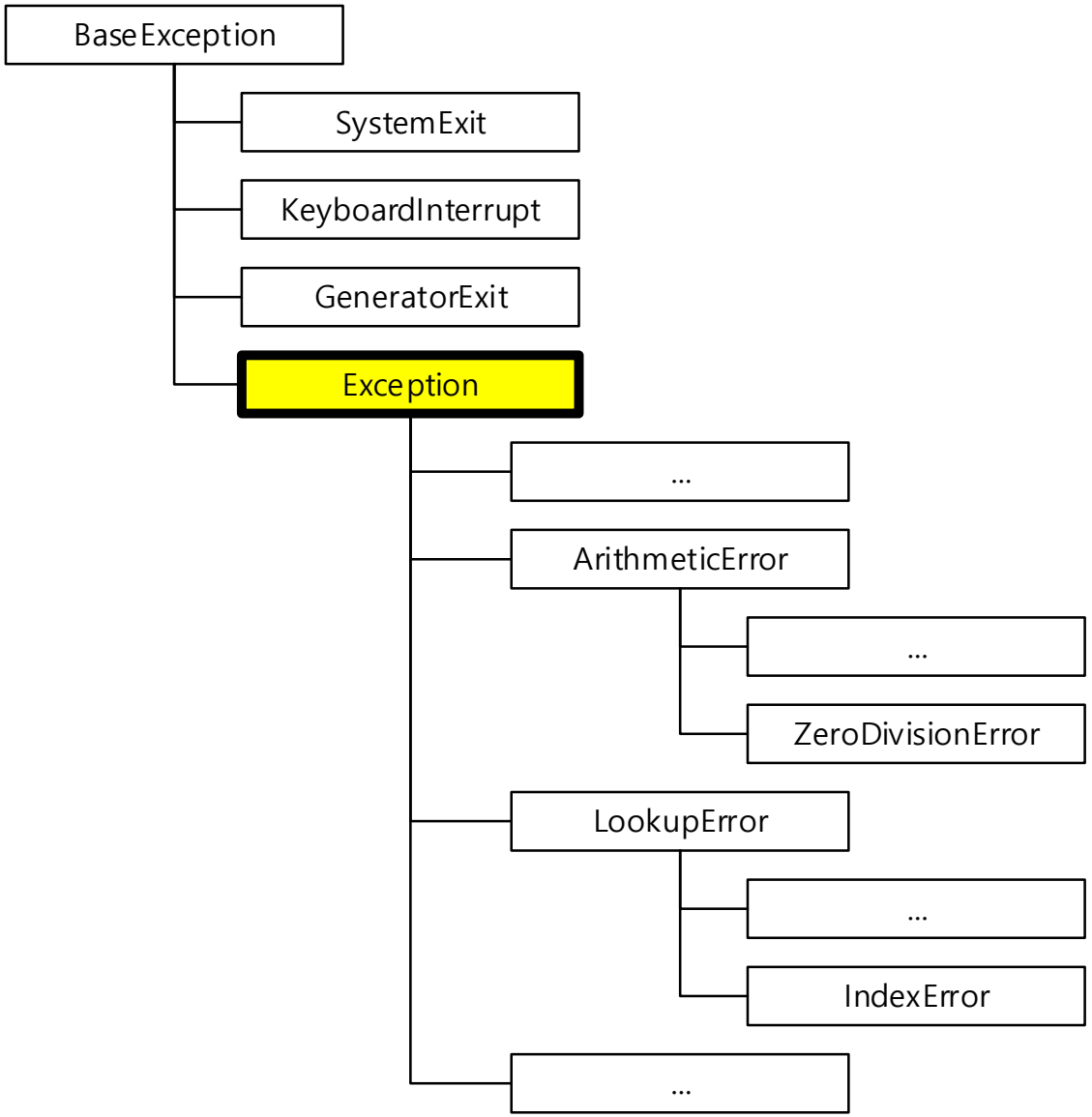
• 실행 결과

```
>try_except_else_finally.py
첨자를 입력하세요:
2
my_list[2]: 3
리스트의 요소 출력에 성공했습니다
.
어떤 일이 있어도 마무리합니다.

>try_except_else_finally.py
첨자를 입력하세요:
10
예외가 발생했습니다 (list index out
of range)
어떤 일이 있어도 마무리합니다.
```

Exception 클래스

❖ 파이썬에서 제공하는 예외 형식들은 거의 모두 Exception 클래스로부터 파생 (미리 정의 되어 있는 표준 라이브러리 사용)



Exception 클래스

❖ 예제 : 10/ignored_exception.py

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

try:
    print("첨자를 입력하세요:")
    index = int(input())
    print(my_list[index]/0)
except Exception as err:
    print("1) 예외가 발생했습니다. ({0})".format(err))
except ZeroDivisionError as err:
    print("2) 0으로 나눌 수 없습니다. ({0})".format(err))
except IndexError as err:
    print("3) 잘못된 첨자입니다. ({0})".format(err))
```

1)

2)

3)

- 실행 결과

```
>ignored_exception.py
첨자를 입력하세요:
10
1) 예외가 발생했습니다. (list index out of range)

>ignored_exception.py
첨자를 입력하세요:
2
1) 예외가 발생했습니다. (division by zero)
```

❖ 위 코드에서는 어떤 경우에도 2)과 3) 예외 처리 구문은 실행할 기회를 얻지 못함.

우리도 예외 좀 일으켜보자

❖ raise문을 이용하면 예외를 직접 일으킬 수 있음.

```
text = input()
if text.isdigit() == False:
    rasie Exception("입력받은 문자열이 숫자로 구성되어 있지 않습니다."):
```

❖ 실습 1 (raise로 예외 일으키기)

다짜고짜 raise문을 통해 예외를 일으킵니다.

```
>>> raise Exception("예외를 일으킵니다.")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    raise Exception("예외를 일으킵니다.")
Exception: 예외를 일으킵니다.
```

예외를 처리하는 곳이 없다 보니 파이썬 인터프리터가 받아 예외 정보를 출력했습니다.

❖ 실습 2 (raise로 일으킨 예외를 try~except로 받기)

```
>>> try:
        raise Exception("예외를 일으킵니다.")
except Exception as err:
        print("예외가 일어났습니다. : {0}".format(err))


예외가 일어났습니다. : 예외를 일으킵니다.
```

우리도 예외 좀 일으켜보자

❖ 예제 : 10/raise_in_function.py

```
def some_function():
    print("1~10 사이의 수를 입력하세요:")
    num = int(input())
    if num < 1 or num > 10:
        raise Exception("유효하지 않은 숫자입니다.: {0}".format(num))
    else:
        print("입력한 수는 {0}입니다.".format(num))

try:
    some_function()
except Exception as err:
    print("예외가 발생했습니다. {0}".format(err))
```

함수 안에서 일어난 예외가 except문으로 처리되지 않으면 함수 밖으로 다시 던져집니다.

- 실행 결과

```
>raise_in_function.py
1~10 사이의 수를 입력하세요:
5
입력한 수는 5입니다.

>raise_in_function.py
1~10 사이의 수를 입력하세요:
12
예외가 발생했습니다. 유효하지 않은 숫자입니다.: 12
```

우리도 예외 좀 일으켜보자

❖ 예제 : 10/raise_again.py

```
def some_function():
    print("1~10 사이의 수를 입력하세요:")
    num = int(input())
    if num < 1 or num > 10:
        raise Exception("유효하지 않은 숫자입니다.: {0}".format(num))
    else:
        print("입력한 수는 {0}입니다.".format(num))

def some_function_caller():
    try:
        some_function()
    except Exception as err:
        print("1) 예외가 발생했습니다. {0}".format(err))
        raise

try:
    some_function_caller()
except Exception as err:
    print("2) 예외가 발생했습니다. {0}".format(err))
```

some_function() 안에서 일으킨 예외가 일단 some_function_caller()의 except 절에서 처리됩니다.

except절에서 다시 raise를 실행함으로써 some_function()에서 올린 예외를 그대로 다시 some_function_caller()의 호출자에게 올립니다.

- 실행 결과

```
>raise_again.py
1~10 사이의 수를 입력하세요:
5
입력한 수는 5입니다.

>raise_again.py
1~10 사이의 수를 입력하세요:
20
1) 예외가 발생했습니다. 유효하지 않은 숫자입니다.: 20
2) 예외가 발생했습니다. 유효하지 않은 숫자입니다.: 20
```

내가 만든 예외 형식

- 파이썬이 제공하는 내장 예외 형식만으로 충분하지 않을 때 직접 예외 클래스를 정의할 수 있음.
- 사용자 정의 예외 클래스는 Exception 클래스를 상속하여 정의함.

```
class MyException(Exception):
    pass
```

- 필요에 따라 다음과 같이 데이터 속성이나 메소드를 추가 가능.

```
class MyException(Exception):
    def __init__(self):
        super().__init__( "MyException이 발생했습니다." )
```

우리도 예외 좀 일으켜보자

❖ 10/InvalidIntException.py

```
class InvalidIntException(Exception):
    def __init__(self, arg):
        super().__init__('정수가 아닙니다.: {0}'.format(arg))

def convert_to_integer(text):
    if text.isdigit(): # 부호(+, -) 처리 못함.
        return int(text)
    else:
        raise InvalidIntException(text)

if __name__ == '__main__':
    try:
        print('숫자를 입력하세요:')
        text = input()
        number = convert_to_integer(text)
    except InvalidIntException as err:
        print('예외가 발생했습니다 ({0})'.format(err))
    else:
        print('정수 형식으로 변환되었습니다 : {0}({1}'.format(number, type(number)))
```

- 실행 결과

```
>InvalidIntException.py
숫자를 입력하세요:
123
정수 형식으로 변환되었습니다 : 123(<class 'int'>

>InvalidIntException.py
숫자를 입력하세요:
abc
예외가 발생했습니다 (정수가 아닙니다.: abc)
```

정리

- 표준 라이브러리(클래스)에 정의 되어 있는 예외는 자동적으로 발생 됨으로, except에 예외형식만 명시 해 주면 됨.

- 표준 라이브러리에 없는 예외는
  1) Exception 클래스를 정보와 함께 직접 발생 시키고, except에 예외형식을 명시 하거나
  2) 예외 라이브러리(클래스)를 Exception 클래스의 파생 클래스로 직접 만들어서 정보와 함께 발생 시키고, except에 예외형식을 명시