



오류를 어떻게 다뤄야 할까

파이썬 프로그래밍

Contents

- ❖ 오류(Error)
- ❖ 예외(Exception)
- ❖ 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



오류의 종류

❖ 오류 (error)

- 구문 오류 (syntax error)
 - 프로그램 실행 전에 발생하는 오류
- 런타임 오류 (runtime error) / 예외 (exception)
 - 프로그램 실행 중에 발생하는 오류

❖ 구문 오류

구문 오류가 발생하는 코드

```
# 프로그램 시작
print("# 프로그램이 시작되었습니다!")

# 구문 오류 발생 코드
print("# 예외를 강제로 발생시켜 볼게요!")
```

→ 닫는 따옴표로 문자열을 닫지 않았습니다.



오류의 종류

❗ 오류

SyntaxError: EOL while scanning string literal

- SyntaxError
- 구문에 문제가 있어 프로그램 실행부터 불가능한 경우

구문 오류 해결

프로그램 시작

```
print("# 프로그램이 시작되었습니다!")
```

구문 오류 발생 코드

```
print("# 예외를 강제로 발생시켜 볼게요!")
```

→ 닫는 따옴표로 문자열을 닫아서 해결합니다.



오류의 종류

❖ 예외 / 런타임 오류

- 실행 중에 발생하는 오류

예외가 발생하는 코드

```
# 프로그램 시작  
print("# 프로그램이 시작되었습니다!")
```

```
# 예외 발생 코드  
list_a[1]
```

프로그램이 시작되었습니다! → 여기까지는 프로그램이 정상으로 실행되었다는 것을 확인할 수 있습니다.

Traceback (most recent call last):

File "test.py", line 5, in <module>

list_a[1]

NameError: name 'list_a' is not defined



오류의 종류

예외 해결

```
# 프로그램 시작
```

```
print("# 프로그램이 시작되었습니다!")
```

```
# 예외 발생 코드 해결
```

```
list_a = [1, 2, 3, 4, 5] → 여러 메시지에서 정의하지 않았다고 하니 정의해 줍니다.
```

```
list_a[1]
```



- ❖ 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'
```



기본 예외 처리

❖ 예외 처리 (exception handling)

- 조건문을 사용하는 방법
 - 기본 예외 처리
- try 구문을 사용하는 방법

❖ 예외 상황 확인하기

예외가 발생할 수 있는 코드

```
# 숫자를 입력받습니다.  
number_input_a = int(input("정수 입력> "))  
  
# 출력합니다.  
print("원의 반지름:", number_input_a)  
print("원의 둘레:", 2 * 3.14 * number_input_a)  
print("원의 넓이:", 3.14 * number_input_a * number_input_a)
```



기본 예외 처리

- 정수를 입력하지 않았을 경우

정수 입력> 7센티미터 → 정수로 변환할 수 없는 문자열을 입력했습니다.

Traceback (most recent call last):

File "test.py", line 2, in <module>

number_input_a = int(input("정수 입력> "))

ValueError: invalid literal for int() with base 10: '7센티미터'



기본 예외 처리

❖ 조건문으로 예외 처리하기

- `isdigit()` 함수 사용하여 숫자로만 구성된 글자인지 확인

```
01  # 숫자를 입력받습니다.
02  user_input_a = input("정수 입력> ")
03
04  # 사용자 입력이 숫자로만 구성되어 있을 때
05  if user_input_a.isdigit():
06      # 숫자로 변환합니다.
07      number_input_a = int(user_input_a)
08      # 출력합니다.
09      print("원의 반지름:", number_input_a)
10      print("원의 둘레:", 2 * 3.14 * number_input_a)
11      print("원의 넓이:", 3.14 * number_input_a * number_input_a)
12  else:
13      print("정수를 입력하지 않았습니다.")
```



기본 예외 처리

- 정수 입력하면 정상적인 값 출력

정수 입력> 8

원의 반지름: 8

원의 둘레: 50.24

원의 넓이: 200.96

- 정수로 변환할 수 없는 문자열 입력하는 경우

정수 입력> yes!!

정수를 입력하지 않았습니다.



try except 구문

❖ try except 구문

- 예외 처리할 수 있는 구문

```
try:
```

예외가 발생할 가능성이 있는 코드

```
except:
```

예외가 발생했을 때 실행할 코드

- 어떤 상황에 예외가 발생하는지 완벽하게 이해하고 있지 않아도 프로그램이 강제로 죽어버리는 상황은 막을 수 있음



try ~ except로 예외 처리하기

- ❖ try 절 안에 문제가 없을 경우의 코드 블록을 배치하고
- ❖ except 절에는 문제가 생겼을 때 뒤처리를 하는 코드 블록 배치

```
try:  
    # 문제가 없을 경우 실행할 코드  
except:  
    # 문제가 생겼을 때 실행할 코드
```

* 어떤 상황에 예외가 발생하는지 완벽하게 이해하고 있지 않아도 프로그램이 강제로 죽어버리는 상황은 막을 수 있음

❖ 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 except 구문

- 예시

```
01  # try except 구문으로 예외를 처리합니다.
02  try:
03      # 숫자로 변환합니다.
04      number_input_a = int(input("정수 입력> ")) → 예외가 발생할 가능성이 있는 구문
05      # 출력합니다.
06      print("원의 반지름:", number_input_a)
07      print("원의 둘레:", 2 * 3.14 * number_input_a)
08      print("원의 넓이:", 3.14 * number_input_a * number_input_a)
09  except:
10      print("무언가 잘못되었습니다.") → 예외가 발생했을 때 실행할 구문
```

정수 입력> yes!!

무언가 잘못되었습니다.



try except 구문

❖ try except 구문과 pass 키워드 조합하기

- 예외가 발생하면 일단 처리해야 하지만, 해당 코드가 딱히 중요한 부분이 아닌 경우 프로그램 강제 종료부터 막는 목적으로 except 구문에 아무 것도 넣지 않고 try 구문 사용
- pass 키워드를 빈 except 구문에 넣음

```
try:  
    예외가 발생할 가능성이 있는 코드  
except:  
    pass
```



try except 구문

- 예시 - 숫자로 변환되는 것들만 리스트에 넣기

```
01  # 변수를 선언합니다.
02  list_input_a = ["52", "273", "32", "스파이", "103"]
03
04  # 반복을 적용합니다.
05  list_number = []
06  for item in list_input_a:
07      # 숫자로 변환해서 리스트에 추가합니다.
08      try:
09          float(item) # 예외가 발생하면 알아서 다음으로 진행은 안 되겠지?
10          list_number.append(item) # 예외 없이 통과했으면 리스트에 넣어줘!
11      except:
12          pass
13
14  # 출력합니다.
15  print("{} 내부에 있는 숫자는".format(list_input_a))
16  print("{}입니다.".format(list_number))
```

실행결과

```
['52', '273', '32', '스파이', '103'] 내부에 있는 숫자는
['52', '273', '32', '103']입니다.
```



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:  
    # 문제가 생겼을 때 실행할 코드  
except 예외형식2:  
    # 문제가 생겼을 때 실행할 코드
```

❖ 예제 : 10/multiple_except.py

• 실행 결과

```
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 일
어납니다.

index가 0~2를 벗어나면 my_list[i
ndex]에서 IndexError가 발생합니
다.

```
>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

```
try:
    # 문제가 없을 경우 실행할 코드
except 예외형식1 as err:
    # 문제가 생겼을 때 실행할 코드
except 예외형식2 as err:
    # 문제가 생겼을 때 실행할 코드
```

❖ 예제 : 10/multiple_except2.py

• 실행 결과

```
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))
```

```
>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

• 실행 결과

```
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.")
```

```
>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 except else 구문

- 예시

```
01  # try except else 구문으로 예외를 처리합니다.  
02  try:  
03      # 숫자로 변환합니다.  
04      number_input_a = int(input("정수 입력> "))  
05  except:  
06      print("정수를 입력하지 않았습니다.")  
07  else:  
08      # 출력합니다.  
09      print("원의 반지름:", number_input_a)  
10      print("원의 둘레:", 2 * 3.14 * number_input_a)  
11      print("원의 넓이:", 3.14 * number_input_a * number_input_a)
```

실행결과 1

정수 입력> 7

원의 반지름: 7

원의 둘레: 43.96

원의 넓이: 153.86

실행결과 2

정수 입력> yes!!

정수를 입력하지 않았습니다.



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

• 실행 결과

```
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.")
```

```
>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:
    # 실행할 코드 블록
except:
    # 예외처리 코드블록
else:
    # except절을 만나지 않았을 경우 실행하는 코드블록
finally:
    # 무조건 실행되는 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.
```

finally 구문

❖ finally 구문

```
01  # try except 구문으로 예외를 처리합니다.  
02  try:  
03      # 숫자로 변환합니다.  
04      number_input_a = int(input("정수 입력> "))  
05      # 출력합니다.  
06      print("원의 반지름:", number_input_a)  
07      print("원의 둘레:", 2 * 3.14 * number_input_a)  
08      print("원의 넓이:", 3.14 * number_input_a * number_input_a)  
09  except:  
10      print("정수를 입력해달라고 했잖아요?!")  
11  else:  
12      print("예외가 발생하지 않았습니다.")  
13  finally:  
14      print("일단 프로그램이 어떻게든 끝났습니다.")
```



finally 구문

```
실행결과 1
정수 입력> 273 [Enter]
원의 반지름: 273
원의 둘레: 1714.44
원의 넓이: 234021.06
예외가 발생하지 않았습니다.
일단 프로그램이 어떻게든 끝났습니다.
```

```
실행결과 2
정수 입력> yes!! [Enter]
정수를 입력하지 않았습니다.
일단 프로그램이 어떻게든 끝났습니다.
```

❖ try, except, finally 구문의 조합

- try 구문은 단독으로 사용할 수 없으며,
반드시 except 구문 또는 finally 구문과 함께 사용해야 함
- else 구문은 반드시 except 구문 뒤에 사용해야 함



finally 구문

- try + except 구문 조합
- try + except + else 구문 조합
- try + except + finally 구문 조합
- try + except + else + finally 구문 조합
- try + except 구문 조합




finally 구문

- 오류 경우

try + else 구문 조합

```
# try except 구문으로 예외를 처리합니다.  
try:  
    # 숫자로 변환합니다.  
    number_input_a = int(input("정수 입력> "))  
    # 출력합니다.  
    print("원의 반지름:", number_input_a)  
    print("원의 둘레:", 2 * 3.14 * number_input_a)  
    print("원의 넓이:", 3.14 * number_input_a * number_input_a)  
else:  
    print("프로그램이 정상적으로 종료되었습니다.")
```

 오류

SyntaxError: Invalid syntax

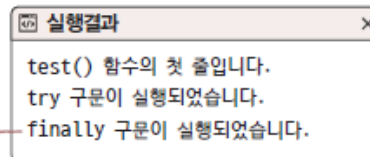


finally 구문

❖ try 구문 내부에서 return 키워드를 사용하는 경우

```
01 # test() 함수를 선언합니다.
02 def test():
03     print("test() 함수의 첫 줄입니다.")
04     try:
05         print("try 구문이 실행되었습니다.")
06         return
07         print("try 구문의 return 키워드 뒤입니다.")
08     except:
09         print("except 구문이 실행되었습니다.")
10     else:
11         print("else 구문이 실행되었습니다.")
12     finally:
13         print("finally 구문이 실행되었습니다.")
14     print("test() 함수의 마지막 줄입니다.")
15
16 # test() 함수를 호출합니다.
17 test()
```

finally 구문은 무조건 실행됩니다. ←



실행결과

test() 함수의 첫 줄입니다.
try 구문이 실행되었습니다.
finally 구문이 실행되었습니다.

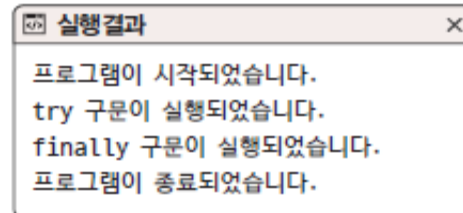
- try 구문 내부에 return 키워드 있음
 - try 구문 중간에서 탈출해도 finally 구문 무조건 실행됨



finally 구문

❖ 반복문과 함께 사용하는 경우

```
01  print("프로그램이 시작되었습니다.")
02
03  while True:
04      try:
05          print("try 구문이 실행되었습니다.")
06          break
07          print("try 구문의 break 키워드 뒤입니다.")
08      except:
09          print("except 구문이 실행되었습니다.")
10      finally:
11          print("finally 구문이 실행되었습니다.")
12          print("while 반복문의 마지막 줄입니다.")
13  print("프로그램이 종료되었습니다.")
```

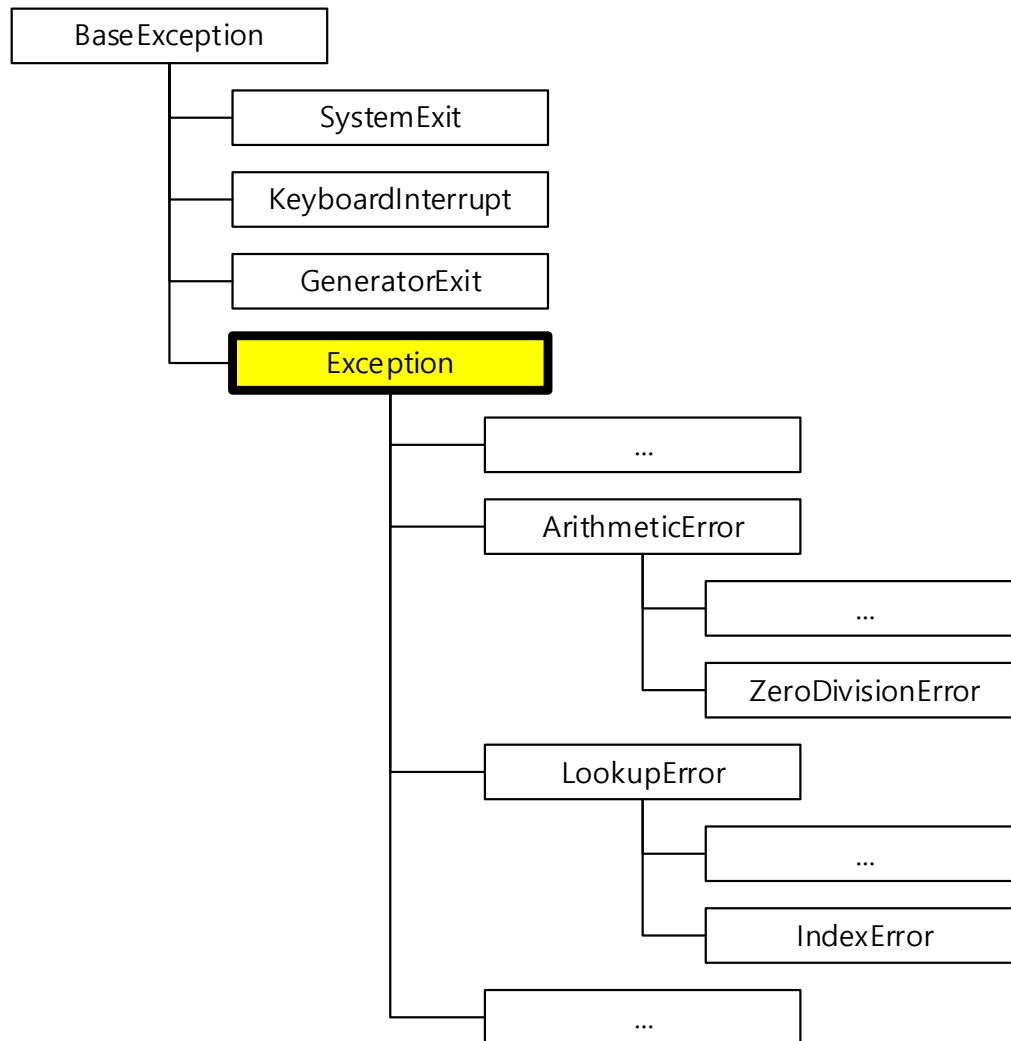


- break 키워드로 try 구문 전체 빠져나가도 finally 구문 실행



Exception 클래스

❖ 파이썬에서 제공하는 예외 형식들은 거의 모두 Exception 클래스로부터 파생



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:           ← 1)
    print("1) 예외가 발생했습니다. ({0})".format(err))
except ZeroDivisionError as err:   ← 2)
    print("2) 0으로 나눌 수 없습니다. ({0})".format(err))
except IndexError as err:          ← 3)
    print("3) 잘못된 첨자입니다. ({0})".format(err))
```

• 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:
    raise Exception("입력받은 문자열이 숫자로 구성되어 있지 않습니다.")
```

❖ 실습 1 (raise로 예외 일으키기)

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

다짜고짜 raise문을 통해 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." )
```



❖ 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)
```

확인문제

- ❖ 리스트 내부에서 특정 값이 어디 있는지 확인할 때는 리스트의 `index()` 함수를 아래처럼 사용합니다.

```
>>> numbers = [52, 273, 32, 103, 90, 10, 275]
>>> numbers.index(52)
0
>>> numbers.index(103)
3
```

- ❖ 해당 값이 여러 개 있을 경우에는 다음과 같이 첫 번째 값의 위치를 리턴합니다.

```
>>> numbers = [1, 1, 1, 1, 1, 1, 1]
>>> numbers.index(1)
0
```



확인문제

- ❖ 그런데 이 함수는 리스트의 없는 값에 접근하려고 할 때 `ValueError` 예외가 발생합니다.

```
>>> numbers = [52, 273, 32, 103, 90, 10, 275]
>>> numbers.index(1000000)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    numbers.index(1000000)
ValueError: 1000000 is not in list
```

- ❖ 다음 코드의 빈칸을 조건문을 사용한 경우와 `try except` 구문을 사용한 경우로 작성하여 예외가 발생하지 않고 코드가 실행결과처럼 출력되게 만들어주세요.



확인문제

```
numbers = [52, 273, 32, 103, 90, 10, 275]
```

```
print("# (1) 요소 내부에 있는 값 찾기")
```

```
print("- {}는 {} 위치에 있습니다.".format(52, numbers.index(52)))
```

```
print()
```

```
print("# (2) 요소 내부에 없는 값 찾기")
```

```
number = 10000
```

①

```
print("- {}는 {} 위치에 있습니다.".format(number, numbers.index(number)))
```

②

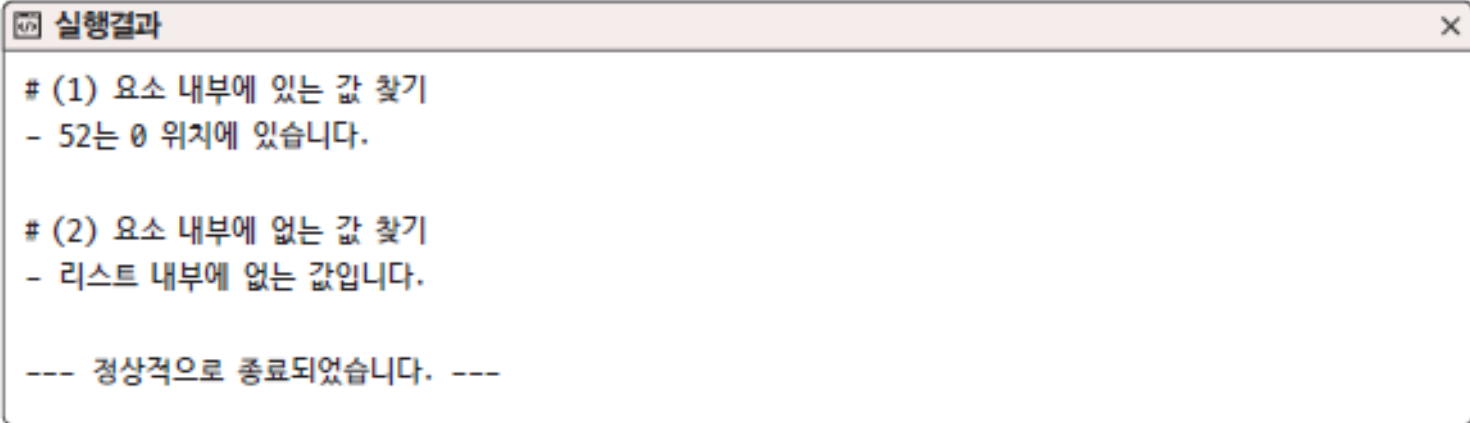
```
print("- 리스트 내부에 없는 값입니다.")
```

```
print()
```

```
print("--- 정상적으로 종료되었습니다. ---")
```



확인문제





Thank You !

파이썬 프로그래밍