



CHAPTER 9

예외 처리

Exception Handling



박진수 교수

서울대학교·경영대학

jinsoo@snu.ac.kr



학습 목차

- 오류와 예외 처리
- try-except-else-finally문
- try-finally문
- raise문

오류와 예외 처리

Errors and Exception Handling





- 프로그램 오류의 종류와 디버깅(debugging)

- 문법 오류(syntax error)

- 오류 수정이 쉬움(즉, 버그 잡기가 쉬움)

- 런타임 오류(runtime error)

- 예외(exception) 처리를 해야 함

- 논리 오류(logical error)

- 오류 없이 프로그램이 실행되지만 예상한 결과가 나오지 않는 경우

- $1 / 2 * 3$

- 문법 오류는 컴파일 단계에서, 런타임 오류와 논리 오류는 실행 단계에서 발생

- 실행 단계에서의 오류

- 문법 오류가 없는 프로그램이라도 실행단계에서 오류가 발생하지 않는다고 말할 수 없다

- 실행 단계에서 오류를 피하기 위해서는 샘플 데이터를 넣어 테스트 해보는 것이 좋다

- 샘플 데이터는 신중히 선택하고 오류 발생 가능한 다양한 경우의 수를 고려 —> be a creative pessimist!

- 발견된 오류로 인해 프로그램 논리를 변경해야 할 수도 있다





- 예상치 못한 값이나 에러가 발생한 경우
- 프로그램 실행 도중에 발생
- 흔치 않은 상황
- 예시

- 잘못된 입력 값을 입력한 경우
- 값을 0으로 나눈 경우
- 숫자로 변환할 수 없는 문자열을 정수로 변환하려고 했을 경우
- 파일 저장 시 디스크 용량이 부족한 경우
- 존재하지 않는 파일 불러오기

```
15 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
f = open('hw1.py')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundException: [Errno 2] No such file or directory:
'hw1.py'
```

- 예외 처리를 하지 않거나 실패하면

- 프로그램이 실행 중 갑작스럽게 종료

- 예외 방지

- 코드 작성시 주의를 기울이면 대부분의 예외 발생을 막을 수 있다

- e.g., 유효한 입력 값 확인

- 추적(traceback)

- 오류 메시지를 통해 예외가 발생한 줄의 번호를 확인할 수 있다

- 발생한 예외의 종류와 예외가 발생한 이유에 대한 간략한 설명이 나타난다

The diagram illustrates a ZeroDivisionError exception. On the left, a screenshot of a terminal shows the error message: "15 / 0", followed by a Traceback with the last call being from standard input (stdin), line 1, in <module>. The specific error is "ZeroDivisionError: division by zero". A yellow arrow points from the word "ZeroDivisionError" in the traceback to the corresponding line in the Python code on the right. The Python code consists of a try block containing a division operation (15 / 0) and an except block that catches ZeroDivisionError, prints the error, and outputs the message "계산할 수 없어요ㅠㅠ".

```
try:  
    15 / 0  
except ZeroDivisionError as err:  
    print(err)  
    print('계산할 수 없어요ㅠㅠ')  
  
division by zero  
계산할 수 없어요ㅠㅠ
```



● 예외 처리(exception handling)

- ⦿ 오류를 처리하는 품격있는 해결책
- ⦿ 현재 실행 중인 코드가 오류를 발견하고 예외를 발생시키면 호출한 명령문에서 처리

● 용어

⦿ **try**문

- ⦿ 오류가 발생할 가능성이 있는 코드를 실행

⦿ 예외 전달(throws an exception)

- ⦿ 오류를 감지한 코드가 예외객체(exception object)를 생성해서 호출한 명령문 또는 운영체제로 넘겨준다

⦿ 예외 처리(catches the exception)

- ⦿ 예외 상황을 처리하는 코드가 예외를 전달받아서 적절한 조치를 취한다



다음 프로그램을 작성해서 실행해보자

```
i = int(input('정수를 입력하세요: '))
print(f'15 / {i} = {15 / i}')
```

정수를 입력하세요: 0

실습

- ➊ 3을 입력해보자.
- ➋ 그 다음 0을 입력해보자
- ➌ 무슨 일이 일어나는가?



다음 프로그램을 작성해서 실행해보자

사용자가 **0**을 입력하면 **ZeroDivisionError**로 예외 처리를 하는 프로그램

```
i = int(input('정수를 입력하세요: '))

try: # 오류가 발생할 가능성이 있는 코드를 try문 안에 넣는다.
    print(f'15 / {i} = {15 / i}')
except ZeroDivisionError:
    print('0으로 나눌 수 없습니다. 다른 숫자를 입력하세요. ')
```

정수를 입력하세요: 0

코드 설명

- ➊ 사용자가 **0**을 입력하면
- ➋ **ZeroDivisionError**로 예외 처리를 한다

실습

- ➊ **0**을 입력해보자
- ➋ 무슨 일이 일어나는가?

그런데 다시 입력하려면 프로그램이 종료되었기 때문에 프로그램을 다시 실행해야 한다



프로그램을 중단하지 않고 예외 처리를 할 수는 없을까?

```
while True:  
    try: # 오류가 발생할 가능성이 있는 코드를 try문 안에 넣는다.  
        i = int(input('정수를 입력하세요: '))  
        print(f'15 / {i} = {15 / i}')  
        break # 유효한 값을 입력했으므로 처리한 후 무한 루프를 빠져나간다.  
    except ZeroDivisionError:  
        print('0으로 나눌 수 없습니다. 다른 숫자를 입력하세요.')
```

정수를 입력하세요: 0

정수를 입력하세요: 5

코드 설명

- 무한 루프를 사용해
- 0을 입력하면 예외 처리를 하고,
- 0이 아닌 숫자를 입력할 때까지
- 다시 입력을 요구한다

실습

- 0을 입력해보자
- 5를 입력해보자
- 만약 숫자가 아닌 문자를 입력하면 어떻게 될까?
- 문자 a를 입력해보자



여러 종류의 예외를 한번에 처리할 수는 없을까?

```
while True:  
    try: # 오류가 발생할 가능성이 있는 코드를 try문 안에 넣는다.  
        i = int(input('정수를 입력하세요: '))  
        print(f'15 / {i} = {15 / i}')  
        break # 유효한 값을 입력했으므로 처리한 후 무한 루프를 빠져나간다.  
    except ZeroDivisionError:  
        print('0으로 나눌 수 없습니다. 다른 숫자를 입력하세요.')  
    except ValueError:  
        print('숫자를 입력해야 합니다.')
```

코드 설명

- ➊ ZeroDivisionError와
- ➋ ValueError를
- ➌ 한꺼번에 처리한다

실습

- ➊ 0을 입력해보자
- ➋ a를 입력해보자
- ➌ 5를 입력해보자



여러 개의 예외를 하나의 그룹으로 묶어 처리할 수도 있다

```
while True:  
    try: # 오류가 발생할 가능성이 있는 코드를 try문 안에 넣는다.  
        i = int(input('정수를 입력하세요: '))  
        print(f'15 / {i} = {15 / i}')  
    break # 유효한 값을 입력했으므로 처리한 후 무한 루프를 빠져나간다.  
except (ZeroDivisionError, ValueError): # 반드시 괄호로 묶어야 한다.  
    print('잘못된 값이 입력되었습니다. 다시 입력하세요.')
```

코드 설명

- ➊ ZeroDivisionError와
- ➋ ValueError를
- ➌ 한꺼번에 처리한다

실습

- ➊ 0을 입력해보자
- ➋ a를 입력해보자
- ➌ 5를 입력해보자



예외 객체를 변수로 참조하면 해당 오류가 무엇인지 확인할 수 있다

```
while True:  
    try: # 오류가 발생할 가능성이 있는 코드를 try문 안에 넣는다.  
        i = int(input('정수를 입력하세요: '))  
        print(f'15 / {i} = {15 / i}')  
    break # 유효한 값을 입력했으므로 처리한 후 무한 루프를 빠져나간다.  
except (ZeroDivisionError, ValueError) as err: # 반드시 괄호로 묶어야 한다.  
    print('다음과 같은 예외가 발생했습니다:', err)  
    print('다시 입력하세요.')
```

코드 설명

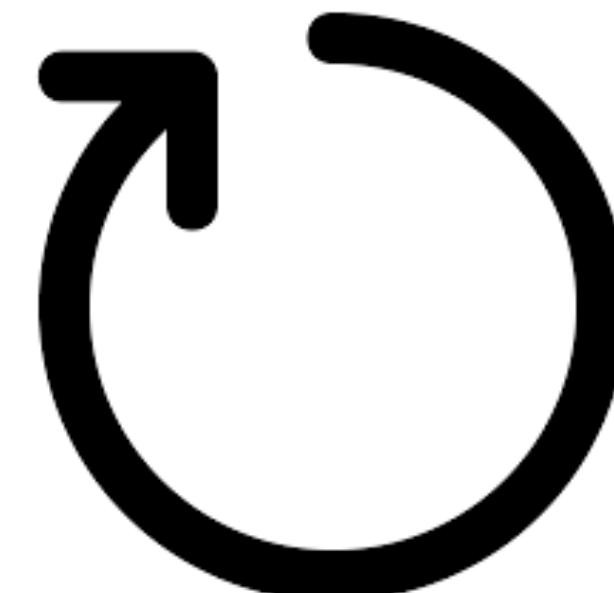
- ➊ ZeroDivisionError와
- ➋ ValueError를
- ➌ 변수 `err`로 참조한다

실습

- ➊ 0을 입력해보자.
- ➋ a를 입력해보자.
- ➌ 5를 입력해보자.

try-except-else-finally 문

try-except-else-finally statement





예외처리 작성 방법

try:

try-명령문

except 예외그룹-1 [as 변수-1]:

예외처리-명령문-1

except 예외그룹-2 [as 변수-2]:

예외처리-명령문-2

...

except 예외그룹-N [as 변수-N]:

예외처리-명령문-N

else: 선택 사항

else-명령문

finally: 선택 사항

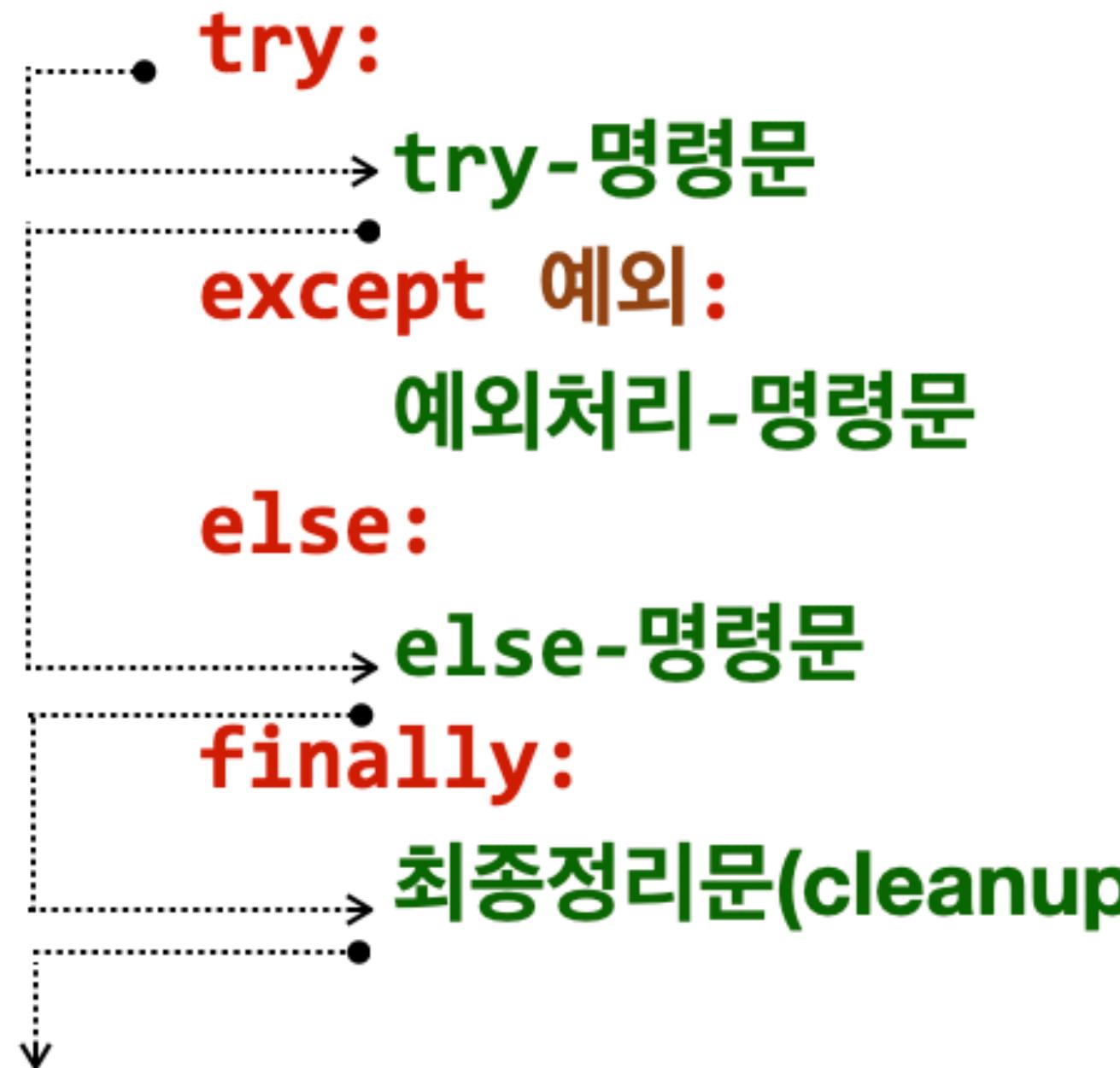
finally-명령문

- **try** 문이 오면 최소한 한 개 이상의 **except** 문이 와야 한다
 - 단, **try ... finally** 문에서는 **except** 문을 사용하지 않는다
- **except** 문은 **try** 문 안의 코드에서 오류가 발생할 때 예외를 처리하는 코드를 포함
- **as 변수-N**은 선택 사항으로 **예외그룹-N** 을 **변수**로 참조할 수 있게 한다
- **try** 문 안의 코드에서 오류가 발생하지 않고 정상적으로 종료하면 **else** 문은 반드시 실행된다
 - 하지만 **try** 문 안의 코드에서 오류가 발생하면 **else** 문은 실행되지 않는다
- **else** 문은 마지막 **except** 문 다음에 와야 한다
 - **else** 문은 선택 사항이라 **try-except** 문 다음에 반드시 사용할 필요는 없다
- **finally** 문은 예외 처리 블록의 맨 마지막에 와야 한다
 - **finally** 문도 선택 사항이라 **try-except** 문 다음에 반드시 사용할 필요는 없다
- **finally** 문이 있다면 **try** 문 안의 코드에서 오류가 발생했는지 여부와 상관 없이 항상 마지막에 실행된다

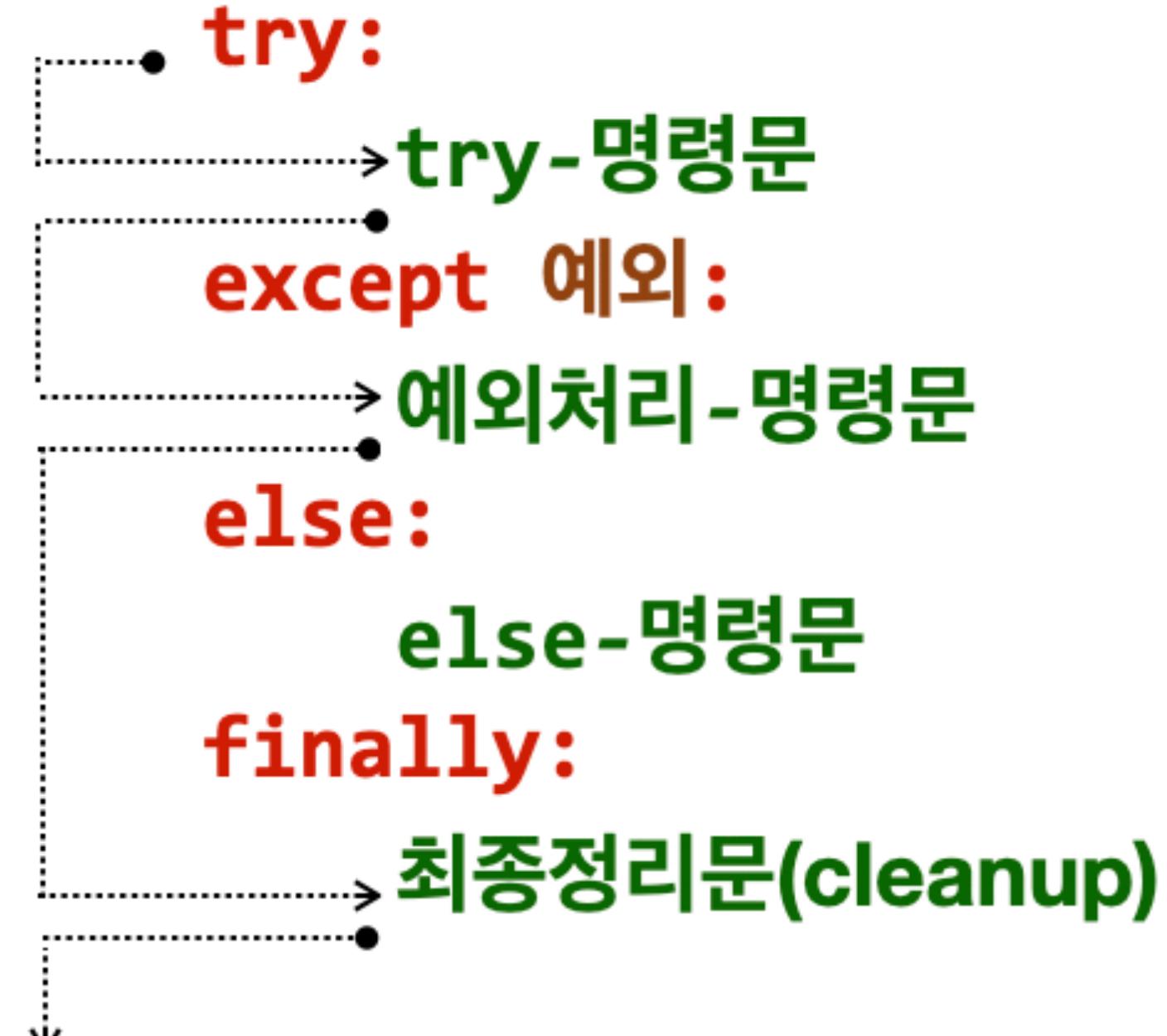


try ... except ... else ... finally 문 처리 순서

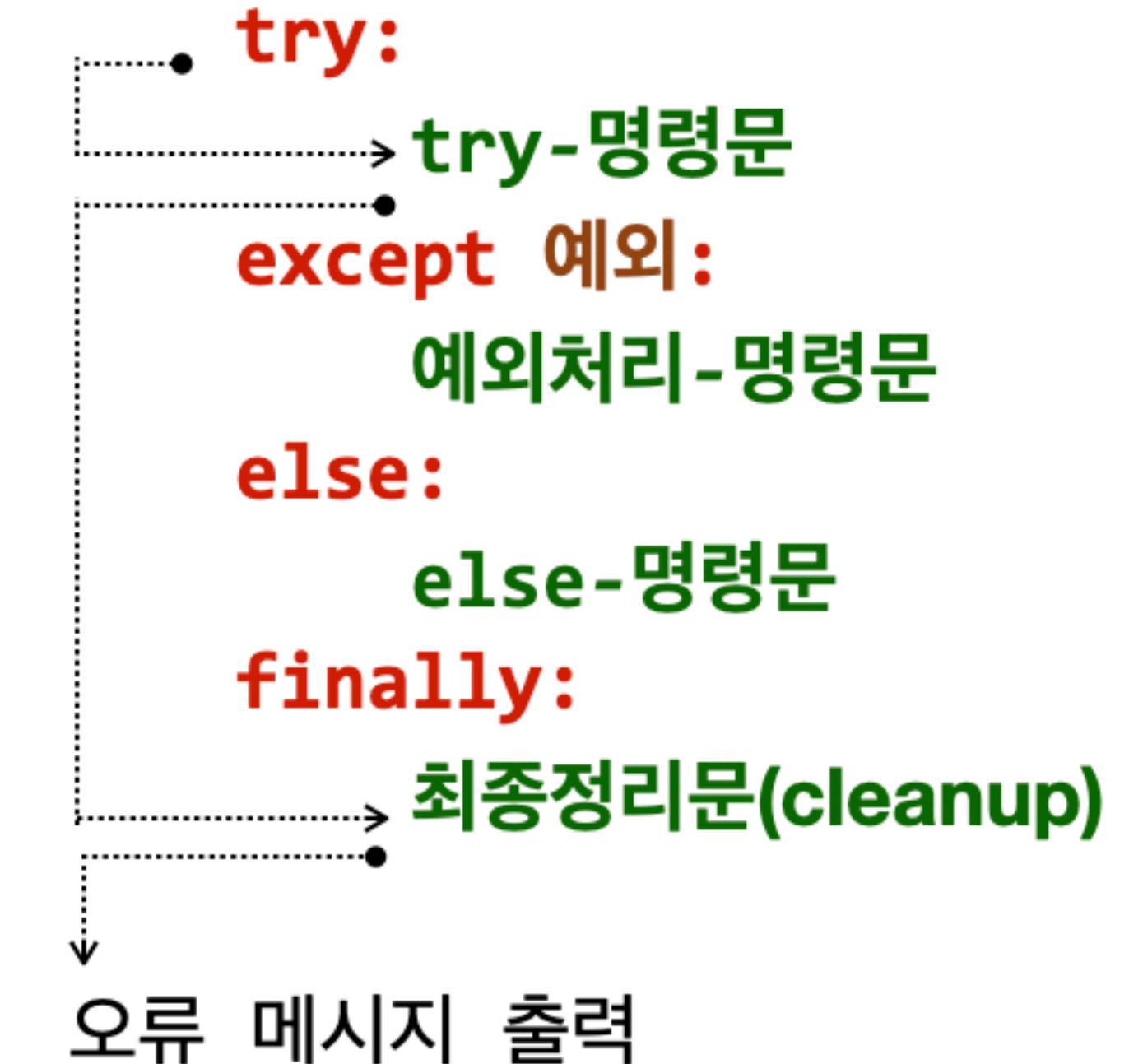
예외없는 정상 종료



예외처리를 할 경우



예외처리를 하지 못한 경우





● 예외객체(exception object)

- 예외가 발생하면 예외객체가 생성된다
- 주로 해당 예외와 관련된 디폴트 오류 메시지를 담고 있다

● 각 **except**문의 예외그룹은

- 하나의 예외를 처리할 수도 있고
 - 예) **except ArithmeticError:**
- 튜플 형태의 예외 목록을 한번에 처리하는 것도 가능
 - 예) **except (ArithmeticError, ValueError, RuntimeError):**

```
try:  
    15 / 0  
except ZeroDivisionError:  
    print('0으로 나눌 수 없습니다.')
```

0으로 나눌 수 없습니다.

```
try:  
    15 / 'a' # 0 외 다른 문자도 입력해본다.  
except (ZeroDivisionError, ValueError, TypeError, RuntimeError):  
    print('뭔가 잘못되었습니다.')
```

뭔가 잘못되었습니다.



as 변수

예외가 하나의 예외객체인 경우와 튜플 형태의 예외그룹인 경우 모두 'as 변수'(선택 사항)를 통해 한 개의 변수로 참조할 수 있는데, 이 경우 해당 변수는

- 발생한 예외객체를 담고 있으며
- 예외처리-명령문에서 사용

예)

- except ArithmeticError as err:
- except (ArithmeticError, ValueError, RuntimeError) as err:

```
try:  
    15 / 0  
except ZeroDivisionError as err:  
    print('다음과 같은 예외가 발생했습니다:', err)
```

다음과 같은 예외가 발생했습니다: division by zero

```
try:  
    15 / 0  
except (ZeroDivisionError, TypeError) as err:  
    print('다음과 같은 예외가 발생했습니다:', err)
```

다음과 같은 예외가 발생했습니다: division by zero



● except:

- 처리할 예외그룹을 정하지 않았을 경우
- 발생하는 모든 예외를 처리하지만 이것은 좋은 방법이 아니니 권장하지 않는다(발생 가능한 오류가 무엇인지 모를 때만 어쩔 수 없이 이 방법을 사용)
- 가급적이면 발생 가능한 모든 오류를 구체적인 예외그룹(하나의 예외객체 또는 여러 목록)을 지정할 것을 권장

```
try:  
    15 / 0 # 문자도 입력해본다.  
except: # 모든 예외를 처리하기 하기 때문에 권장하지 않는 방법  
    print('뭔가 잘못된 것 같아요 ㅠㅠ')
```

뭔가 잘못된 것 같아요 ㅠㅠ

```
try:  
    15 / 0 # 문자도 입력해본다.  
except Exception as err: # 모든 예외를 처리하기 때문에 권장하지 않는 방법  
    print(err) # Exception보다 ZeroDivisionError가 더 바람직하다
```

division by zero



except문 작성 순서

● except문 실행 순서

- try문에 예외가 발생하면 각 except문은 순서대로 실행된다
- 발생한 예외와 일치하는 except문에 도달하면 해당 except문의 예외처리-명령문이 실행됨
- 발생한 예외와 except문에 작성한 예외가 일치하는 경우는 다음과 같다
 - 발생한 예외가 예외그룹에 있는 예외와 같은 타입인 경우
 - 발생한 예외가 예외그룹에 있는 예외의 하위 클래스인 경우

● except문 작성 팁

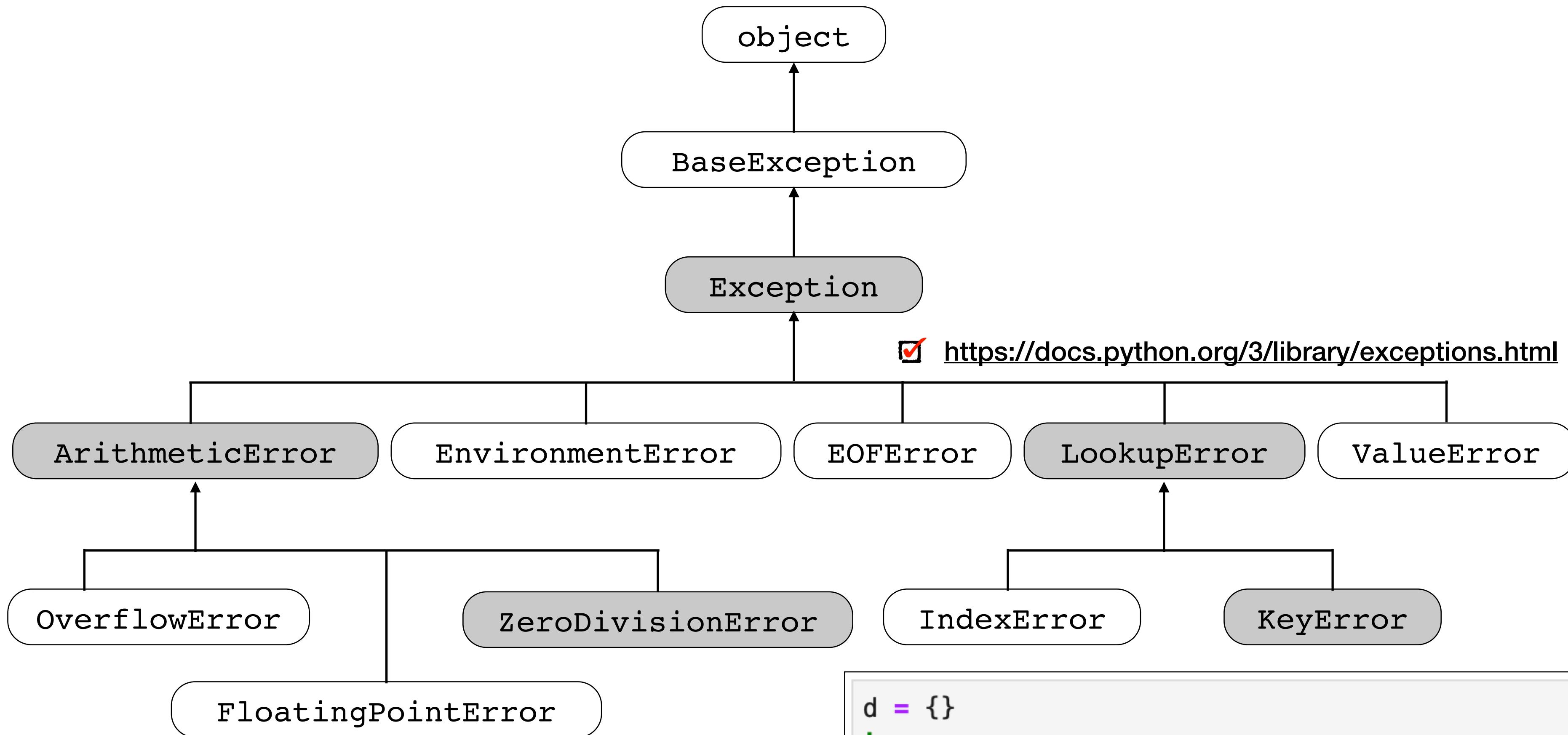
- 세부적인 예외 클래스(종류)를 먼저 작성하고 좀 더 범위가 넓은 일반적인 예외 클래스를 작성해야 한다

```
try:  
    15 / 0  
except ArithmeticError as err:  
    print('산술 예외가 발생했습니다:', err)  
except ZeroDivisionError:  
    print('0으로 나눌 수 없습니다:', err)
```

```
try:  
    15 / 0  
except ZeroDivisionError as err:  
    print('0으로 나눌 수 없습니다:', err)  
except ArithmeticError as err:  
    print('산술 예외가 발생했습니다:', err)
```



예시 : 예외 클래스 계층



<https://docs.python.org/3/library/exceptions.html>

```
d = {}
try:
    x = d[5]
except LookupError:      # 순서가 잘못됨
    print('LookupError 예외가 발생했습니다.')
except KeyError:        # KeyError가 LookupError보다 위로 가야함
    print('딕셔너리 키(key) 값을 찾을 수 없습니다.')
```



● **pass** 명령어 사용

- 어떤 상황에서는 오류 발생 시 처리를 하지 않고 그냥 통과시켜야 하는 경우도 있다
- 이런 경우 **except**문에 예외 처리 코드를 넣지 않고 그냥 **pass** 명령어를 추가하면 된다
- 실행할 명령문에 **pass** 명령어를 넣으면 아무것도 실행하지 않고 넘어간다

```
try:  
    15 / 0  
except ZeroDivisionError:  
    pass
```



아무 것도 일어나지 않는다

try-finally 문

try-finally statement





try ... finally 문

작성 방법

```
try:  
    try-명령문  
finally:  
    finally-명령문
```

- 오류가 발생해 작업을 처리하지 못하는 경우에도,
반드시 실행해야 하는 코드가 있을 때 사용
- 컴퓨터나 프로그램이 충돌이 일어나 갑자기 멈추지 않는 한, **try** 문 안의 오류 발생과 상관없이 **finally** 문은 마지막에 반드시 실행된다

```
try:  
    15 / 0  
finally:  
    print('finally문은 예외가 발생해도 실행된다!!!')
```



예시 : try ... finally 문

```
import http.client
try:
    conn = http.client.HTTPSConnection('www.python.org')
    conn.request('GET', '/')
    response = conn.getresponse()
    print(response.getheader('Connection')) # 'Connection' 헤더 값을 출력한다.
    print(response.status, response.reason) # http 요청에 서버가 응답한 상 태 코드와 원인 문구를 출력한다.
finally:
    conn.close() # 서버 연결을 끊는다.

response.closed # 서버 연결이 끊어졌는지 여부를 확인한다.
```

raise 문

raise statement





오류를 일부러 발생시켜야 하는 경우

`raise` 명령어를 이용해서 예외를 강제로 발생시킨다

```
def marry(girl, boy): # 결혼 함수를 정의한다
    raise NotImplementedError('marry() 함수는 다음 버전에서 구현될 예정입니다.')
```

```
# 예외 처리를 한 경우
try:
    marry('네오', '프로도')
except NotImplementedError as err:
    print(err)
```

```
# 예외 처리를 하지 않은 경우
marry('네오', '프로도')
```



NotImplementedError

NotImplementedError를 사용해서 꼭 구현해야할 기능을 아직 구현하지 않았을 경우 고의로 예외를 발생시킬 수 있다

```
class Ontology: pass

o1 = Ontology()
o2 = Ontology()

def merge(ontology1, ontology2):
    raise NotImplementedError('merge() 함수는 다음 버전에서 구현될 예정입니다. ')

try:
    merge(o1, o2)
except NotImplementedError as e:
    print(e)
pass
```



예외처리 작성 방법

raise 예외클래스[(전달인자)] from 근원예외객체

```
i = 0
if i == 0:
    raise ValueError # ValueError()와 같다.
```

```
i = 0
if i == 0:
    raise ValueError('0이 입력되었습니다.')
```

```
i = 0
if i == 0:
    raise ValueError(-3.14)
```

- 예외클래스는 두 종류가 있다
 - 파이썬이 기본으로 제공하는 예외 클래스
 - 사용자가 직접 정의하는 예외 클래스
 - * 사용자 정의 예외 클래스는 반드시 **Exception** 클래스를 상속해야 한다
 - * 클래스 상속에 관한 설명은 12장에서 자세히 다룬다
- 전달인자가 주어지면 예외를 처리할 때 전달인자의 값을 출력
 - 따라서 주로 해당 예외 클래스와 관련된 오류 메시지를 전달인자로 사용
 - 전달인자는 선택 사항이라 생략 가능
- 근원예외객체를 지정하면 근원예외객체부터 예외클래스까지 연결된 연쇄 예외 객체를 생성
 - 근원예외객체는 선택 사항이라 생략 가능
- **raise ... from** 문은 주로 **except** 문 안에서 사용



raise 예외클래스[(전달인자)] **from** 근원예외객체

```
i = 0
try:
    result = 5 / i
except Exception as err:
    raise RuntimeError('뭔가 잘못된 것 같아요 ㅠㅠ') from err
```



raise 문 다음에 예외클래스를 지정하지 않고 **raise** 문만 사용하면, 현재 발생한 오류의 예외를 처리하지 않고 호출한 상위 명령문으로 그 오류를 넘긴다

```
numbers = [7, -5, 8, 3, 'a', 9]
total = 0
try:
    for i in numbers:
        try:
            total += i
        except TypeError:
            print('예외가 발생했으니, 호출한 명령문에서 처리해주세요.')
            raise # 예외 처리를 여기서 하지 않고 호출한 명령문으로 예외를 넘긴다.
except Exception:
    print('숫자만 처리 가능합니다!!!!')
else:
    print(total)
```



raise 문을 일반 명령문에 사용하면 오류가 없어도(현재 발생한 오류가 없어도) **RuntimeError** 예외를 발생시키기 때문에 주의해서 사용해야 한다

```
total = 0
for i in range(1, 11): # 1에서 10까지 합을 구한다.
    total += i
    print(total)
    if total > 20:
        raise # 현재 발생한 예외가 없어도 RuntimeError 예외를 발생시킨다.
```

예외 처리

Lab Exercises





Lab : try-except-else문



● 입력한 값을 정수로 변환하기

- 사용자가 입력한 값을 정수로 형변환을 시도
- 이 때 예외가 발생하면 '형식이 올바르지 않습니다!!!'라는 문구를 출력하고,
- 예외가 발생하지 않으면 입력한 값을 출력

● 실행 결과 예시

```
자연수를 입력하세요....: a  
형식이 올바르지 않습니다!!!
```

```
자연수를 입력하세요....: 5.0  
형식이 올바르지 않습니다!!!
```

```
자연수를 입력하세요....: 5  
5
```



● 무한 루프를 사용하여 사용자가 잘못 입력한 내용을 예외 처리하기

- 예외 처리를 통해 원하는 값의 범위를 입력하지 않으면 계속해서 입력을 요청하는 프로그램을 작성
- input** 함수를 사용해서 사용자가 입력한 값이 1~9 사이의 정수가 아니면 계속해서 입력을 요청
- 정확한 범위의 값을 입력하면 '통과하셨습니다.'라는 문자열을 출력하고 프로그램을 종료

● 실행 결과 예시

```
1~9 사이의 숫자를 입력하세요....: apple
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: 블루베리
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: 0
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: 10
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: -1
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: 1.234
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: 5.0
ValueError: 1~9 사이의 자연수를 입력하세요!!!
1~9 사이의 숫자를 입력하세요....: 5
통과하셨습니다.
```