Exception Handling

Exception Handling

- 오류
  - ✓ 오류의 종류
    - 컴파일 에러(error): 문법적인 오류로 실행이 되지 않는 오류
    - 예외(exception): 실행 도중 외부 요인이나 잘못된 입력 등으로 발생하는 오류
    - 논리적 에러: 런타임 시에 에러가 발생하지는 않고 정상적으로 실행이 되었지만 의도하지 않은 결과가 나오는 경우
    - 단언(assert): 개발자가 의도적으로 특정 조건을 만족했을 때 만 프로그램이 정상적으로 동작하도록 만드는 것
  - ✓ 오류 발생 시 조치
    - 컴파일 에러가 발생하면 에러를 수정해야 실행 됨
    - 컴파일 에러는 IDE를 사용하면 명확하게 표시를 해주므로 비교적 찾기 쉬움
    - 런타임 에러나 논리적 에러는 프로그램 외부의 문제인지 논리적인 오류인지 여러 가지 상황을 고려해야 하므로 에러의 원인을 찾기 어려운 경우가 많음 – 디버깅 수행

Exception Handling

- Debugging
  - ✓ 예외나 논리적인 오류의 원인을 밝혀내고 수정하는 과정
  - ✓ 디버깅 방법
    - 로깅(Logging)
    - IDE가 제공해주는 Debugging Tool을 사용
    - 파일이나 데이터베이스에 로그를 남기는 Logging Tool을 사용
    - 테스트를 자동으로 수행해주는 테스팅 툴을 사용

Exception Handling

- Exception
  - ✓ 문법적으로는 이상이 없어서 컴파일 시점에는 아무런 문제가 없지만 프로그램 실행 중에 발생하는 예기치 않은 사건으로 인해 프로그램이 중단되는 에러
  - ✓ 기대하지 않은 동작(프로그램의 버그 나 이른바 오류 등을 포함)이 발생한 것
  - ✓ 예외가 발생하는 경우
    - 정수를 0으로 나누는 경우
    - 배열의 첨자가 범위를 벗어나거나 음수를 사용하는 경우
    - 부적절한 형 변환이 일어나는 경우
    - 입출력을 위한 파일이 없는 경우 등
  - ✓ 예외 처리의 용도
    - 정상 종료
    - 예외내용 기록
    - 예외 발생 시 무시하고 계속 실행
    - 정상적인 값으로 변경해서 실행

Exception Handling

- Exception
  - ✓ 예외 발생

```
def ten_div(x):
    return 10 / x

print(ten_div(2)) #정상 수행
print(ten_div(0)) #예외 발생
```

```
5.0
ZeroDivisionError                         Traceback (most recent call last)
/var/folders/42/fxl3_n31121f16yj69fbrb_m0000gn/T/ipykernel_6419/2929335998.py in
<module>
      3
      4 print(ten_div(2)) #정상 수행
----> 5 print(ten_div(0)) #예외 발생
/var/folders/42/fxl3_n31121f16yj69fbrb_m0000gn/T/ipykernel_6419/2929335998.py in
ten_div(x)
      1 def ten_div(x):
----> 2     return 10 / x
      3
      4 print(ten_div(2)) #정상 수행
      5 print(ten_div(0)) #예외 발생

ZeroDivisionError: division by zero
```

Exception Handling

- Exception Handling
  - ✓ try ~ except 를 이용한 예외 처리

```
try:
        실행할 코드
except:
        예외가 발생했을 때 처리하는 코드
```

Exception Handling

- Exception Handling
  - ✓ try ~ except 를 이용한 예외 처리

```
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    y = 10 / x
    print(y)
except:    # 예외가 발생했을 때 실행됨
    print('예외가 발생했습니다.')
```

```
나눌 숫자를 입력하세요:  0
예외가 발생했습니다.
```

Exception Handling

- Exception Handling
  - ✓ try ~ except 를 이용한 예외 처리
    - 특정 예외만 처리하고자 하는 경우에는 except 다음에 예외 클래스 이름을 설정하면 됨
    - 특정 예외를 처리하는 경우는 except 를 여러 개 작성해서 예외 종류 별로 처리하는 것이 가능

```
y = [10, 20, 30]

try:
    index, x = map(int, input('인덱스와 나눌 숫자를 입력하세요: ').split())
    print(y[index] / x)
except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
except IndexError:           # 범위를 벗어난 인덱스에 접근하여 에러가 발생했을 때 실행됨
    print('잘못된 인덱스입니다.')
```

Exception Handling

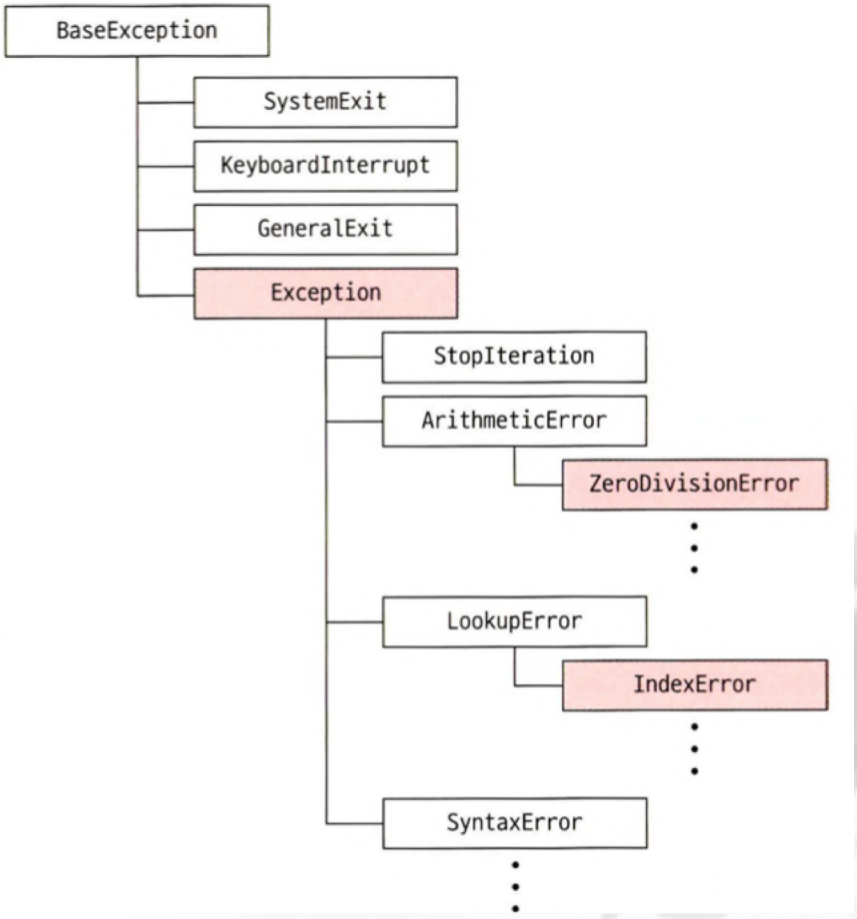
- Exception Handling
  - ✓ try ~ except 를 이용한 예외 처리
    - 에러 메시지를 처리하고자 하는 경우에는 예외 클래스 다음에 as 변수명 을 설정해서 출력하면 가능

```
y = [10, 20, 30]

try:
    index, x = map(int, input('인덱스와 나눌 숫자를 입력하세요: ').split())
    print(y[index] / x)
except ZeroDivisionError as e:                    # as 뒤에 변수를 지정하면 에러를 받아옴
    print('숫자를 0으로 나눌 수 없습니다.', e)     # e에 저장된 에러 메시지 출력
except IndexError as e:
    print('잘못된 인덱스입니다.', e)
```

Exception Handling

- Exception Handling
  - ✓ 예외 클래스 계층: https://docs.python.org/3/library/exceptions.html



Exception Handling

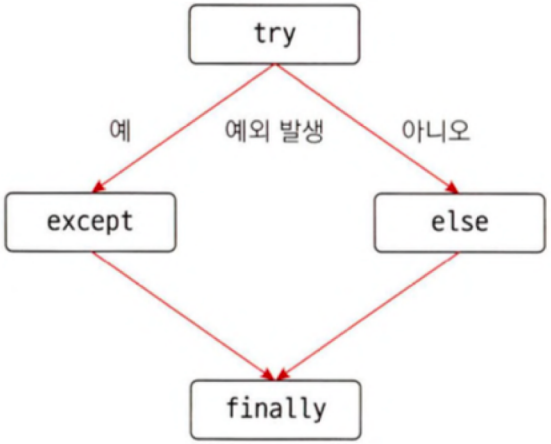
- Exception Handling
  - ✓ else 절에 예외가 발생하지 않을 때 수행할 코드를 작성할 수 있음
  - ✓ finally는 예외 발생 여부에 상관없이 수행되는 코드를 작성할 수 있음

```
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    y = 10 / x
except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
else:                        # try의 코드에서 예외가 발생하지 않았을 때 실행됨
    print(y)
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    y = 10 / x
except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
else:                        # try의 코드에서 예외가 발생하지 않았을 때 실행됨
    print(y)
```

```
나눌 숫자를 입력하세요:  8
1.25
```

Exception Handling

- Exception Handling
  - ✓ else ~ finally



Exception Handling

- Exception Handling
  - ✓ else ~ finally

```
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    y = 10 / x
except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
else:                        # try의 코드에서 예외가 발생하지 않았을 때 실행됨
    print(y)
finally:                     # 예외 발생 여부와 상관없이 항상 실행됨
    print('코드 실행이 끝났습니다.')
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    y = 10 / x
except ZeroDivisionError:    # 숫자를 0으로 나눠서 에러가 발생했을 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
else:                        # try의 코드에서 예외가 발생하지 않았을 때 실행됨
    print(y)
finally:                     # 예외 발생 여부와 상관없이 항상 실행됨
    print('코드 실행이 끝났습니다.')
```

Exception Handling

- Exception Handling
  - ✓ else ~ finally

```
나눌 숫자를 입력하세요:  1
10.0
코드 실행이 끝났습니다.
```

Exception Handling

- Exception Handling
  - ✓ 예외를 강제로 발생시키기: raise 예외(에러 메시지)

```
try:
    x = int(input('3의 배수를 입력하세요: '))
    if x % 3 != 0:                                  # x가 3의 배수가 아니면
        raise Exception('3의 배수가 아닙니다.')    # 예외를 발생시킴
    print(x)
except Exception as e:                              # 예외가 발생했을 때 실행됨
    print('예외가 발생했습니다.', e)
```

Exception Handling

- Exception Handling
  - ✓ 현재 예외를 다시 발생시키기

```
def three_multiple():
    try:
        x = int(input('3의 배수를 입력하세요: '))
        if x % 3 != 0:                                  # x가 3의 배수가 아니면
            raise Exception('3의 배수가 아닙니다.')    # 예외를 발생시킴
        print(x)
    except Exception as e:                              # 함수 안에서 예외를 처리함
        print('three_multiple 함수에서 예외가 발생했습니다.', e)
        raise    # raise로 현재 예외를 다시 발생시켜서 상위 코드 블록으로 넘김

try:
    three_multiple()
except Exception as e:                                  # 하위 코드 블록에서 예외가 발생해도 실
행됨
    print('스크립트 파일에서 예외가 발생했습니다.', e)
```

Exception Handling

- Exception Handling
  - ✓ Assertion
    - assert는 지정된 조건식이 거짓일 때 AssertionError 예외를 발생시키며 조건식이 참이면 그냥 넘어감
    - assert는 나와서는 안 되는 조건을 검사 할 때 사용
      - assert 조건식
      - assert 조건식, 에러메시지
    - assert는 디버깅 모드에서만 실행되므로 파이썬은 기본적으로 디버깅 모드이며 (__debug__의 값이 True) assert가 실행되지 않게 하려면 python에 -O 옵션을 붙여서 실행(영문 대문자)

```
x = int(input('3의 배수를 입력하세요: '))
assert x % 3 == 0, '3의 배수가 아닙니다.'    # 3의 배수가 아니면 예외 발생, 3의 배수이면 그냥 넘어감
print(x)
```

Exception Handling

- Exception Handling
  - ✓ 사용자 정의 예외를 만들고자 하는 경우에는 예외 클래스를 상속 받아서 상위 클래스의 초기화 메서드를 호출할 때 예외 메시지를 넘겨주면 됨

```
class NotThreeMultipleError(Exception):    # Exception을 상속받아서 새로운 예외를 만듦
    def __init__(self):
        super().__init__('3의 배수가 아닙니다.')

def three_multiple():
    try:
        x = int(input('3의 배수를 입력하세요: '))
        if x % 3 != 0:                      # x가 3의 배수가 아니면
            raise NotThreeMultipleError     # NotThreeMultipleError 예외를 발생시킴
        print(x)
    except Exception as e:
        print('예외가 발생했습니다.', e)

three_multiple()
```