[예외 처리]

- 예외(exception), 오류
- 오류의 종류
- 예외 처리 프로그래밍 방법
- raise 문
- assert 문

예외(exception)

- 프로그램이 수행될 때 예상하지 못한 상황이 발생하는 것
- 예측 가능한 예외 : 발생 여부를 개발자가 사전에 알 수 있는 예외
  - 개발자는 예외를 예측하여 예외가 발생할 때는 어떻게 대응하라고 할 수 있다. 예를 들어 사용자 입력란에 값이 잘못 입력되면, 사용자에게 잘못 입력하였다고 알려주는 방법으로 쉽게 대응 가능
- 예측 불가능한 예외 : 발생 여부를 개발자가 사전에 알기 힘든 예외
  - 리스트의 인덱스 범위를 넘어서는 연산이나, 예상하지 못하게 0으로 나누는 연산 등을 할 경우 오류가 발생하는 경우
  - 예측 불가능한 예외가 발생할 경우, 인터프리터가 자동으로 이런 예외가 발생해 프로그램이 종료한다고 사용자에게 알려주도록 프로그램이 작성되면 좋다.
- 예외 처리 프로그래밍은 프로그램의 완성도를 높이는 데 매우 중요

프로그램의 실행 도중에 나타나는 오류

- ZeroDivisionError는 0으로 나눌 때
- IndexError는 리스트의 범위를 벗어난 동작을 할 때
- FileNotFoundError는 없는 파일을 열려고 할 때
- ValueError는 원하는 값을 입력받지 못할 때
- NameError는 정의되지 않은 변수를 사용할 때
- TypeError는 데이터 형이 맞지 않는 연산을 할 때(예: 문자열에 정수 더하기 등)

오류의 체험

```
>>> 5/0
ZeroDivisionError: division by zero
>>> a=[0,1,2]
>>> a[3]
IndexError: list index out of range
>>> fp=open("Nofile.txt","r")
FileNotFoundError: [Errno 2] No such file or directory: 'Nofile.txt'
>>> int(input('Type float number or string : '))
Type float number or string : 1.23
ValueError: invalid literal for int() with base 10: '1.23'
>>> 1 + var*2
NameError: name 'var' is not defined
>>> "str"+1
TypeError: can only concatenate str (not "int") to str
```

예외 처리 프로그래밍 방법

- 예외 처리의 기본 문법은 다음과 같은 try-except문

```
try :
    [실행 코드] # 예외가 발생할 수도 있는 문장
except : # 예외가 발생하면
    [예외 처리 코드] # 예외를 처리
```

- 실행 코드에서 오류가 발생하지 않으면 예외 처리 코드 부분은 실행되지 않고, 오류가 발생하면 예외 처리 코드 부분을 실행
- 예외 처리라 하지만 대부분의 경우 오류를 수정하기 보다는 오류가 발생했다고 알려주는 정도

예외의 구체적 내용을 알고 있으면?

- except 다음에 다음과 같이 예외의 이름을 명시해 작성 가능

```
try :
    [실행 코드] # 예외가 발생할 수도 있는 문장
except [예외1] : # 예외1이 발생하면
    [예외 처리 코드] # 예외를 처리
```

- 예외가 여러 종류 있으면 except를 여러 번 사용 가능

종합정리

- try-except-else문은 해당 예외가 발생하지 않을 경우 실행할 코드를 else문에 작성
- try-except-finally문은 오류가 있든 없든 실행하는 코드 부분
- 이상의 내용을 하나로 알기 쉽게 요약

```
try :
    [무조건 실행 코드] # 예외가 발생할 수도 있는 문장
except [예외1] : # 예외1 발생하면
    [예외1 처리 코드] # 예외1 발생시 처리
except [예외2] (as 오류메시지) : # 예외2가 발생하면 오류메시지로 기록하고
    [예외2 처리 코드2] # 예외2의 오류메시지 알림
except : # 위에 명시되지 않은 오류이면
    [그 이외의 모든 예외 처리 코드] # 그 이외의 오류 처리
else : #오류가 없으면 다음줄로 가
    [정상 실행 코드] # 이 부분 실행
finally :
    [모든 처리 코드] # 예외가 있을 때, 없을 때 모두 실행되는 부분
```

예제 _

```
def menu():
    print("0. normal")
    print("1. 0으로 나누기")
    print("2. 리스트 인덱스")
    print("3. 없는 파일 열기")
    print("4. 원하는 값 안주기")
    print("5. TypeError")
try:
    menu()
    n = int(input("Select your choice: "))
    if n == 0:
        pass
    elif n == 1:
        a = 5 / 0
    elif n == 2:
        b = [0,1,2]
        b[3]
    elif n == 3:
        open("File.txt","r")
    elif n == 4:
        int(input("Type abc:"))
    elif n == 5:
        "str"+1
except ZeroDivisionError:
    print('0으로 나누기 에러 발생')
except IndexError as e2:
    print(e2)
except FileNotFoundError as e3:
    print(e3)
except (ValueError, TypeError) as e4:
    print(e4)
except:
    print("어떤 에러 발생")
else:
    print("에러 없음")
finally:
    print("예외처리 끝")
```

raise 문

- raise문은 필요할 때 예외를 발생시키는 문장
- 다음은 원의 넓이를 계산하는 간단한 프로그램인데,

```
def area(radius):
    if radius < 0:
        raise RuntimeError("negative radius")
    else:
        return 3.14*radius*radius

print(area(-2))
```

- 만약 반지름을 음수로 주어 계산하면 다음과 같은 결과 얻음

```
 File "raise.py", line 3, in area
    raise RuntimeError("negative radius")
RuntimeError: negative radius
```

assert 문

- assert문은 예외 정보가 조건을 충족하지 않을 경우, 예외를 발생시킴
- assert문과 같이 True 또는 False의 반환이 가능한 함수를 사용. 이 함수가 False를 반환하면 예외가 발생

```
def isEven(n):
    if n % 2 == 0:
        print(n, "is even")
        return True
    else:
        return False

assert isEven(8)
assert isEven(7)
```

- 홀짝을 판단하는 간단한 함수를 만들고 숫자 8과 7을 넘겨 예외 발생을 확인해 보자