예외 처리(Exception Handling)

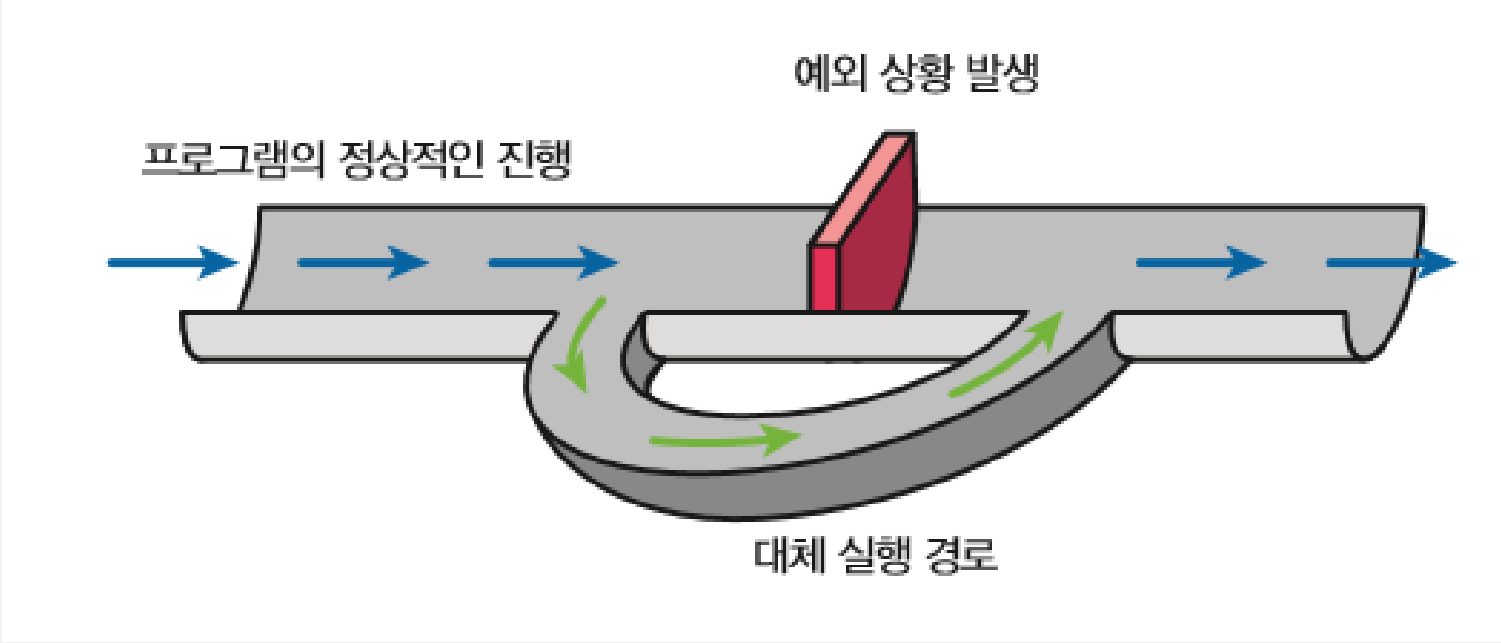
By 윤명근 / 박수현

수업목표

- 예외 처리 필요성
- 예외 처리 문법
- Python 제공 예외 처리

예외 처리의 개념

- 오류가 발생 했을 때 오류를 사용자에게 알려주고 모든 데이터를 저장하게 한 후에 사용자가 우아하게(gracefully) program을 종료할 수 있도록 하는 것이 바람직



예외 처리 필요성

- Program 수행 도중 발생하는 예외 상황 처리
  - 0으로 나누는 경우 (division by 0)
  - 존재하지 않는 파일을 읽으려는 경우
  - list, tuple등 나열형 data의 index 범위 밖을 접근하는 경우

    (ex) list index out of range)
- 예외 처리를 하지 않으면 전체 program이 갑자기 종료 또는 비정상 동작하게 됨

예외 처리

- 오류 !

```
>>> (x, y) = (2, 0)
>>> z = x / y
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    z = x / y
ZeroDivisionError: division by zero
>>>
```

Python에서의 예외 처리

전체적인 구조

```
try :
    예외가 발생할 수 있는 문장
except 오류내용 :
    예외를 처리하는 문장
```

예제

exception_1.py

File Edit Format Run Options Window Help

```
1  (x,y) = (2,0)
2  print('₩n1) x =', x, ', y = ', y)
3
4  try:
5      print('₩n2) try 입구')
6      z = y/x
7      print('3) z = y/x  : ', z)
8
9      print('₩n4) Divided by zero 에러 발생')
10     z = x/y
11
12     print('5) try 출구')
13
14 except ZeroDivisionError as e:
15     print ("₩n6) e =", e)
16     print ("7) 0으로 나누는 예외 발생₩n")
17
18
```

```
1) x = 2 , y =  0

2) try 입구
3) z = y/x  :  0.0

4) Divided by zero 에러 발생

6) e = division by zero
7) 0으로 나누는 예외 발생
```

예외 처리 문법

- try except

```
try:
    코드 블록
except [예외타입 [as 예외변수]]
    예외 처리 코드
[else:
    예외가 발생하지 않은 경우 수행할 코드 블록
finally:
    예외가 발생하든 하지 않든 try 블록 이후 수행할 코드블록]
```

  - except문 처리 방식
    - except 예외타입:
      - 특정 타입 의 예외를 처리하는 경우
    - except:
      - 모든 타입의 예외를 처리하는 경우

11example1.py

File Edit Format Run Options Window Help

```
1  def divide(m, n):
2      try:
3          print("d-1) in divide()")
4          result = m/n
5      except ZeroDivisionError:
6          print("d-2) 0으로 나눌 수 없습니다.")
7      except:
8          print("d-3) ZeroDivisionError 이외의 예외 발생")
9      else:
10         print("d-4) result = ", result)
11         return result
12     finally:
13         print ("d-5) finally 부분의 반드시 실행되는 부분입니다.")
14         print ("d-6) 앞부분에 return 문이 있어도 실행됩니다.")
15
16 #main
17 print("1) main ")
18
19 if __name__ == "__main__":
20     print("₩n2) call divide(3,2)")
21     res = divide(3, 2)
22     print("3) res = ",res)
23
24     print("₩n4) call divide(10,0)")
25     res = divide(10, 0)
26     print("5) res = ",res)
27
28     print("₩n6) call divide(200,3)")
29     res = divide(200, 3)
30     print("7) res = ",res)
31
```

```
1) main

2) call divide(3,2)
d-1) in divide()
d-4) result =  1.5
d-5) finally 부분의 반드시 실행되는 부분입니다.
d-6) 앞부분에 return 문이 있어도 실행됩니다.
3) res =  1.5

4) call divide(10,0)
d-1) in divide()
d-2) 0으로 나눌 수 없습니다.
d-5) finally 부분의 반드시 실행되는 부분입니다.
d-6) 앞부분에 return 문이 있어도 실행됩니다.
5) res =  None

6) call divide(200,3)
d-1) in divide()
d-4) result =  66.66666666666667
d-5) finally 부분의 반드시 실행되는 부분입니다.
d-6) 앞부분에 return 문이 있어도 실행됩니다.
7) res =  66.66666666666667
```

exception_2.py

File Edit Format Run Options Window Help

```
class MyError(Exception):

    def __init__(self, msg):
        self.msg = msg
        print("init) self.msg = ", self.msg)

    def __str__(self):
        print("₩nstr) self.msg = ", self.msg)
        return self.msg


def say_nick(nick):
    print("s-1) in say_nick()")

    if nick == "바보":
            print("s-2) raise MyError()")
            raise MyError("허용되지 않은 별명입니다.")

    print("s-3) nick = ", nick)


#main
print("₩n1) main. ")

if __name__ == "__main__":

    try:
        print("₩n2) call say_nick(₩"천사₩")")
        say_nick("천사")

        print("₩n3) call say_nick(₩"바보₩")")
        say_nick("바보")

    except MyError as e:
        print("₩n4) 사용자가 정의한 Exception handler, Myerror 객체호출")
        print("5) e = ", e)

    finally:
        print ("₩n6) finally 부분으로  반드시 실행되는 부분입니다.")

```

```
1) main.

2) call say_nick("천사")
s-1) in say_nick()
s-3) nick =  천사

3) call say_nick("바보")
s-1) in say_nick()
s-2) raise MyError()
init) self.msg =  허용되지 않은 별명입니다.

4) 사용자가 정의한 Exception handler, Myerror 객체호출
5) e =
str) self.msg =  허용되지 않은 별명입니다.
허용되지 않은 별명입니다.

6) finally 부분으로  반드시 실행되는 부분입니다.
```

polymorphism-1.py

File Edit Format Run Options Window Help

```python
class Vehicle:
    def __init__(self, in_name):
        self.name = in_name
        print("v-int) self.name :", self.name)

    def drive(self):
        raise NotImplementedError("Vehicle class내의 drive()는 추상 method입니다. ")

    def stop(self):
        raise NotImplementedError("Vehicle class내의 stop()은 추상 method입니다. ")

class Car(Vehicle):
    def drive(self):
        return '자동차를 운전합니다. '

    def stop(self):
        return '자동차를 정지합니다. '

class Truck(Vehicle):
    def drive(self):
        return '트럭을 운전합니다. '

    def stop(self):
        return '트럭을 정지합니다. '

#main
print("1) mainWn")
cars = [Truck('현대트럭'),  Car('기아자동차'), Vehicle('No-brand')]

print("Wn2) cars : ", cars, "Wn")

try:
    for car in cars:
        print(">>", car.name + ' : ' + car.drive())

except NotImplementedError as e:
    print("Wn3) NotImplementedError 발생, e :", e)

try:
    for car in cars:
        print(">>", car.name + ' : ' + car.stop())

except :
    print("Wn4) NotImplementedError를 지정하지 않았습니다. ")
```

```
1) main

v-int) self.name : 현대트럭
v-int) self.name : 기아자동차
v-int) self.name : No-brand

2) cars :  [<__main__.Truck object at 0x000001557F6C8E10>, <__main__.Car object
at 0x000001557F4551D0>, <__main__.Vehicle object at 0x000001557F455450>]

>> 현대트럭 : 트럭을 운전합니다.
>> 기아자동차 : 자동차를 운전합니다.

3) NotImplementedError 발생, e : Vehicle class내의 drive()는 추상 method입니다.
>> 현대트럭 : 트럭을 정지합니다.
>> 기아자동차 : 자동차를 정지합니다.

4) NotImplementedError를 지정하지 않았습니다.
```

Python 제공 예외 처리

- https://docs.python.org/ko/3/library/exceptions.html#exception-hierarchy

예외 계층 구조

내장 예외의 클래스 계층 구조는 다음과 같습니다:

```
BaseException
 ├── BaseExceptionGroup
 ├── GeneratorExit
 ├── KeyboardInterrupt
 ├── SystemExit
 └── Exception
      ├── ArithmeticError
      │    ├── FloatingPointError
      │    ├── OverflowError
      │    └── ZeroDivisionError
      ├── AssertionError
      ├── AttributeError
      ├── BufferError
      ├── EOFError
      ├── ExceptionGroup [BaseExceptionGroup]
      ├── ImportError
      │    └── ModuleNotFoundError
      ├── LookupError
      │    ├── IndexError
      │    └── KeyError
      ├── MemoryError
      ├── NameError
      │    └── UnboundLocalError
      ├── OSError
      │    ├── BlockingIOError
      │    ├── ChildProcessError
      │    ├── ConnectionError
      │    │    ├── BrokenPipeError
      │    │    ├── ConnectionAbortedError
      │    │    ├── ConnectionRefusedError
      │    │    └── ConnectionResetError
      │    ├── FileExistsError
      │    ├── FileNotFoundError
      │    ├── InterruptedError
      │    ├── IsADirectoryError
      │    ├── NotADirectoryError
      │    ├── PermissionError
      │    ├── ProcessLookupError
      │    └── TimeoutError
      ├── ReferenceError
      ├── RuntimeError
      │    ├── NotImplementedError
      │    └── RecursionError
      ├── StopAsyncIteration
      ├── StopIteration
      ├── SyntaxError
      │    └── IndentationError
      │         └── TabError
      ├── SystemError
      ├── TypeError
      ├── ValueError
      │    └── UnicodeError
      │         ├── UnicodeDecodeError
      │         ├── UnicodeEncodeError
      │         └── UnicodeTranslateError
      └── Warning
           ├── BytesWarning
           ├── DeprecationWarning
           ├── EncodingWarning
           ├── FutureWarning
           ├── ImportWarning
           ├── PendingDeprecationWarning
           ├── ResourceWarning
           ├── RuntimeWarning
           ├── SyntaxWarning
           ├── UnicodeWarning
           └── UserWarning
```