UNISTUDY

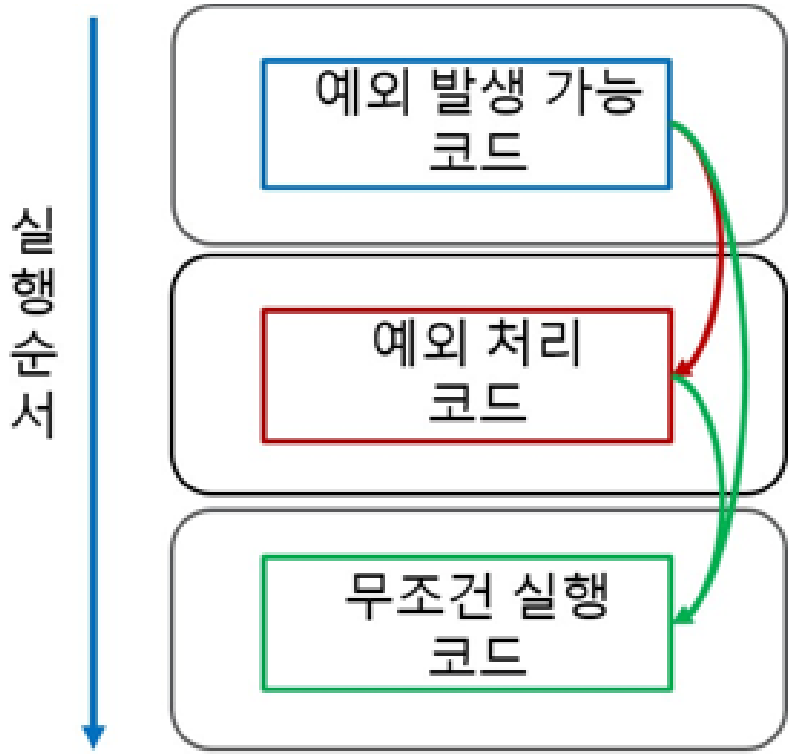
파이썬

27강. 예외처리

1. 예외처리

- 프로그램 코딩 과정에서 작성된 명령어나 명령문의 형식에는 오류가 없지만 프로그램 실행 시점(run time)에서 발생되는 오류를 예외라고 한다. 예외가 발생하면 예외가 발생된 위치에서 프로그램이 종료 된다.
- 예를 들면 뉴스(news)를 제공하는 웹 사이트에서 특정 기간 동안 10페이지를 단위로 뉴스를 수집하 려고 한다. 이를 구현하기 위해서는 각 날짜별로 1~10페이지를 반복하여 뉴스를 수집하는 로직을 작 성하면 된다.
- 그런데 특정 날짜에는 뉴스가 적어서 5페이지 밖에 없다면 6페이지 수집과정에서 예외가 발생하고 프로그램이 중단된다. 따라서 지금까지 수집한 자료는 모두 소멸된다. 결국 이 로직은 무용지 물이 된다.
- 이와 같은 예외를 처리해 주는 과정을 예외처리라고 한다. 즉 프로그램이 처리되는 동안 특정한 문제가 일어났을 때 프로그램이 중단되지 않도록 특별한 처리를 말한다.

1. 예외처리



1. 예외처리

- 파이썬에서는 예외처리 프로그램 로직을 작성할 수 있도록 다음과 같은 형식으로 try ~ except ~ finally 블록을 제공한다. try 블록은 예외가 발생할 가능성이 있는 코드를 작성하는 영역이고, except 블록은 예외가 발생할 경우 예외를 적절하게 처리하는 코드를 작성하는 영역이다. 그리고 finally 블록은 예외와 상관없이 무조건 처리할 코드를 작성하는 영역으로 생략이 가능하다.

```
try :
    예외발생 코드
except 예외처리 클래스 as 변수:
    예외처리 코드
finally :
    항상 실행 코드
```

2. 간단한 예외처리

- 반복 가능한 원소를 갖는 리스트를 대상으로 각 변량에 제곱을 계산하는 예외처리 프로그램 로직을 적 용하는 사례를 예문으로 알아본다. 다음과 같은 x를 대상으로 각 원소에 제곱을 계산할 경우 4번째 원 소('num')는 숫자가 아니기 때문에 계산과정에서 예외가 발생되어 프로그램이 종료된다. 따라서 5번 째 원소 이후부터는 계산결과를 확인할 수 있다.

- x = [10, 30, 25.2, 'num', 14, 51]

- 다음 예문은 예외처리 로직을 적용하여 중단 없이 마지막 원소까지 각 변량에 제곱을 계산하는 예문이다.

2. 간단한 예외처리

chapter08.lecture.step01_try_except.py

```
# (1) 예외 발생 코드

print('프로그램 시작 !!!')
x = [10, 30, 25.2,'num', 14, 51]

for i in x :
    print(i)
     y = i**2   # 예외 발생
     print('y =', y)

print('프로그램 종료')
```

Python Console

```
프로그램 시작 !!!
10
y = 100
30
y = 900
25.2
y = 635.04
num
Traceback (most recent call
last):
  File "<input>", line 6, in
<module>
TypeError: unsupported oper
and type(s) for ** or pow():
'str' and 'int'
```

```
# (2) 예외 처리 코드
print('프로그램 시작 !!!')
```

```
프로그램 시작 !!!
```

2. 간단한 예외처리

```
for i in x :
    try : # try 블록
        y = i**2  # 예외 발생
        print('i=', i, ', y =', y)
    except :  # except 블록
        print('숫자 아님 : ', i)

print('프로그램 종료')
```

```
i= 10 , y = 100
i= 30 , y = 900
i= 25.2 , y = 635.04
숫자 아님 :  num
i= 14 , y = 196
i= 51 , y = 2601
프로그램 종료
```

2. 간단한 예외처리

- 간단한 예외처리 예
- # (1) 예외 발생 코드
- x의 원소 중에서 10, 30, 25.2 원소를 대상으로 제곱(i**2)을 계산하여 결과가 출력된다. 하지만 4번째 원소 'num'을 대상으로 제곱을 계산할 때 예외가 발생하고, 예외 메시지(TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int')를 출력한 다음 프로그램이 중단된다. 문자열 원소를 대상으로 제곱을 계산할 수 없기 때문이다.
- # (2) 예외 처리 코드
- try 블록에 예외가 발생할 수 있는 코드를 작성하고 except 블록에는 예외가 발생할 경우 처리할 코드를 작성한다. for문에서 x의 원소를 i변수에 하나씩 넘기면서 각 변량에 제곱을 계산하는 과정에 서 예외가 발생하면 except 블록에서 작성한 print문에 의해서 '숫자 아님'과 i의 값을 출력하고, 프로그램 중단 없이 for문으로 이동하여 x의 다음 원소를 넘기고 제곱을 계산한다.
- for문의 반복이 종료되면 프로그램의 마지막 명령문이 실행되어 '프로그램 종료'가 출력되고, 프로그 램이 종료된다.

3. 다중 예외처리

- try 블록에서 여러 유형의 예외가 발생될 경우 각 유형별로 예외를 처리하기 위해서 여러 개의 except 블록을 지정할 수 있다. except 명령어 형식은 다음과 같이 except 명령어 다음에 예외를 처리할 수 있는 클래스와 as 다음에 예외 정보를 저장할 수 있는 참조변수를 지정할 수 있다.

- except 예외처리 클래스 as 변수:
- 예외처리 클래스는 builtins 모듈에서 제공된다. 프롬프트에서 다음과 같이 dir 내장함수를 이용하 여 builtins 모듈의 예외처리 클래스를 확인할 수 있다.

3. 다중 예외처리

```
>>>import builtins
>>>dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', '
BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'Child
ProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRef
usedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ell
ipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'File
NotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'I
OError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError',
'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None',
'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError',
'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessL
ookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'Runti
meError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'Syntax
Error', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'Timeout
Error', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'U
nicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarni
ng', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivision
Error' 생략 ...]
```

3. 다중 예외처리

- 산술적인 예외(ArithmeticError), 파일 열기 예외(FileNotFoundError), 기타 예 외(Exception)를 다중으로 처리하는 예문이다.

chapter08.lecture.step01_try_except.py

```
# 유형별 예외처리
print('\n유형별 예외처리 ')
try :
    div = 1000 / 2.53
    print('div = %5.2f' %(div))  # 정상
    div = 1000 / 0          # 1차: 산술적 예외

    f = open('c:\\test.txt')  # 2차: 파일 열기
    num = int(input('숫자 입력 : ')) # 3차: 기타 예외
    print('num =', num)
# 다중 예외처리 클래스
except ZeroDivisionError as e:  # 산술적 예외처리
    print('오류 정보 :', e)
except FileNotFoundError as e: # 파일 열기 예외처리
    print('오류 정보 : ', e)
except Exception as e :  # 기타 예외 처리
```

Python Console

```
유형별 예외처리
div = 395.26
오류 정보 : division by zero
finally 영역 - 항상 실행되는 영역


유형별 예외처리
div = 395.26
오류 정보 : [Errno 2] No such file or directory: 'c:\\test.txt'
finally 영역 - 항상 실행되는 영역

유형별 예외처리
div = 395.26
```

3. 다중 예외처리

```
    print('오류 정보 : ', e)

finally :
    print('finally 영역 - 항상 실행되는
    영역')
```

```
숫자 입력 : >? '123'
오류 정보 :  invalid literal f
or int() with base 10: "'12
3'"
finally 영역 - 항상 실행되는 영역
```

3. 다중 예외처리

- 중첩 예외처리 예

- # 1차: 산술적 예외
- try 블록에서 분모를 0으로 나누는 명령문에서 최초로 산술적인 예외가 발생한다. 이러한 산술적인 예외는 ZeroDivisionError 클래스를 이용하여 예외를 처리한다. as 다음에 오는 참조변수 e는 해당 예외정보를 넘겨받는 변수이다. 따라서 변수를 출력하면 어떤 예외가 발생되어서 예외가 처리되 는지를 확인할 수 있다. 예외처리 결과는 '오류정보'가 출력되고, finally블록의 명령문이 실행된 후 프로그램이 종료된다.
- # 2차: 파일 열기
- 2차 예외 처리 결과를 확인하기 위해서는 1차 산술적인 예외 코드(div = 1000 / 0) 앞부분에 #을 붙여서 주석 처리해야 한다. 해당 경로에 'test.txt' 파일이 존재하는 경우에는 오류가 발생되지 않지만 해당 경로에 파일이 없는 경우에는 예외가 처리된다. 예외처리 결과는 '오류정보'가 출력되고, finally 블록의 명령문이 실행된 후 프로그램이 종료된다.

3. 다중 예외처리

- # 3차: 기타 예외
- 3차 예외 처리 결과를 확인하기 위해서는 1차, 2차 예외 코드 앞부분에 #을 붙여서 주석 처리해야 한다. 그리고 input()함수 실행 과정에서 숫자가 아닌 문자열을 입력하면 예외가 처리된다. 예외처리 결과는 '10진수로 유효하지 않은 문자'라는 오류정보가 출력되고, finally 블록의 명령문이 실행된 후 프로그램이 종료된다.
- 이때 예외처리 클래스는 Exception 클래스를 이용하였다. Exception 클래스는 사용자가 예외의 유형을 정확히 판단할 수 없는 경우 사용할 수 있는 클래스이다. 특히 Exception 클래스는 모든 예외를 처리할 수 있는 클래스로 예외 처리 클래스 중에서 최상 클래스에 해당된다. 따라서 except 블록에서 맨 마지막에 이 클래스를 적용하여 예외를 처리해야 한다. 그렇지 않으면 구문에러 (SyntaxError)가 발생한다.

THANK YOU