Python 예외 처리

키워드 : Python 예외 (https://docs.python.org/3.7/reference/executionmodel.html#exceptions), Python 예외 처리 (https://docs.python.org/3.7/reference/compound_stmts.html#try), Python 예외 발생 (https://docs.python.org/3.7/reference/simple_stmts.html#raise)

우리가 프로그래밍 코드를 작성할 때, 에러나 오류를 마주하는 것은 필수 불가결한 것입니다. 다행스럽게도 에러 발생 자체를 막을 수는 없지만, 에러 발생 시 특정 행동을 취하도록 코드를 작성할 수 있습니다. 때로는, 특별한 이유에 의해서 오류를 의도적으로 발생시키기도 합니다.

이번 장에서는 코드 작성 시 마주하게 되는 에러를 처리하는 방법에 대해서 정리해보도록 하겠습니다.

예외(Exceptions)

먼저, 강의에서 프로그램의 오류가 발생하는 이유는 아래와 같이 크게 두 가지로 나눌 수 있다고 말씀드렸습니다.

> 문법 에러 vs 예외

문법 에러는 개발자가 파이썬 문법에 어긋나게 코드를 작성 했을 경우 발생하는 오류입니다.

```
def sum(x, y):
return x + y # IdentationError 에러 발생!
```

위 코드는 들여 쓰기가 적절히 이루어지지 않았기 때문에 에러가 발생합니다. 문법 에러의 경우엔 코드 구문이 실행되기 전에 검사하며, 에러 발생 시 프로그램 코드가 실행되지 않습니다.

다음은, 예외입니다. 예외는 프로그램 코드 실행 중에 오류 또는 기타 예외적인 조건을 마주했을 경우에 발생하하게 됩니다. 프로그램 코드 실행 중에 발생할 수 있는 예외의 종류는 생각보다 굉장히 많습니다. 그렇다면, 예외를 어떻게 처리할 수 있는지 계속해서 살펴보도록 하겠습니다.

try ~ except

예외를 처리하기 위한 가장 기본적인 방법은 이번 절에서 다룰 `try ~ except` 예외 처리 구문을 이용하는 것입니다. 예외 처리 구문은 코드 블록의 제어를 통해 정상적인 흐름을 벗어나는 수단이며, 예외가 감지된 지점에서 직, 간접적으로 호출 한 예외 처리 코드 블록에 의해 프로그램 오류를 처리할 수 있습니다.

이게 어떤 의미인지 코드와 함께 알아보도록 하겠습니다.

```
def sum():
    i = 0
    try:
        # 1번 코드, 예외 발생 가능성이 있는 코드 구문
        while i < 100:
            i += 1
    except:
        # 2번 코드, 예외 발생시 실행되는 코드 구문
        i = 0
    return i

print(sum()) # 100
```

위 코드 예시는 i의 값을 100이 될 때까지 더하고 반환하는 간단한 반복문 코드입니다. 위 예시 코드는 아무런 예외가 발생하지 않지만, 예외가 발생할 수도 있다고 가정하고 기본 구조를 살펴보겠습니다.

우선, try 구문의 코드 블록인 1번 코드는 예외가 발생할 가능성이 있는 코드 구문을 작성합니다. 코드 구문을 위에서부터 한 줄 한 줄 실행하는 도중에 예외가 발생하면, 예외가 발생하는 시점에 코드가 더 이상 실행되지 않고, 2번 코드 구문인 except 구문의 코드 블록을 실행합니다.

앞서 언급했듯이, except 구문의 코드는 예외가 발생했을 경우에 실행하고 싶은 코드 구문을 작성합니다. 단, try ~ except 구문 작성 시에 except 코드 블록에 아무런 코드도 작성하지 않으면(처리할 코드를 작성하지 않으면), 에러가 발생한다는 점, 함께 기억해주세요!

```
1 + '1' # 1번 코드, TypeError
1 / 0 # 2번 코드, ZeroDivisionError
i + 1 # 3번 코드, NameError
```

위 코드는 파이썬 코드 실행 중에 발생할 수 있는 예외 중에서 간단한 기본적인 예시입니다. 하나하나 살펴보도록 하겠습니다.

1. 1번 코드는 숫자 1과 문자 '1'을 더하는 연산을 할 경우에 발생하는 TypeError의 예입니다.
   (자바스크립트에서는 가능하지만, 파이썬 언어에서는 지원하지 않습니다.)
2. 2번 코드는 숫자 1을 0으로 나누려고 할 경우에 발생하는 ZeroDivisionError입니다.
3. 3번 코드는 아직 선언되지 않은, 변수 i에 곧 바로 1을 더하는 연산을 실행할 때 발생하는 NameError입니다.
   (초기화 하는 작업이 필요합니다. e.g., i = 0)

파이썬 언어에는 여러 가지 예외가 존재한다고 말씀드렸습니다. 첫 번째로 살펴본 except 구문의 경우엔 어떤 예외가 발생할지 모르지만, 모든 예외에 대한 처리가 가능하게 하고 싶을 경우에 사용합니다.

반대로 우리가 코드 작성 시에 어떤 예외가 발생할지 알 순 없지만, 필요에 의해서 특정 예외가 발생할 경우에만 처리하고 싶은 경우가 있을 수 있습니다. 그럴 경우에 에러의 이름을 특정지어서 예외 처리 구문을 작성할 수 있습니다.

```
try:
    1 + '1'
except TypeError:
    print('Type Error!')
```

위 코드와 같이 except 구문 다음에 처리할 예외를 특정해서 작성할 수 있습니다. 만약, 복수 개의 에러를 특정 지어 처리하고 싶을 경우엔, 아래 코드와 같이 except 구문을 나열하여 작성할 수 있습니다.

```
try:
    i + 1 # NameError
    '1' + 1  # TypeError
except TypeError:
    print('Type Error')
except NameError:
    print('Name Error')

# Name Error 출력
```

위 코드에서 한 가지 눈여겨볼 점은 except 구문 작성 순서에 상관없이, try 블록을 안에 i + 1 코드 구문에서 가장 먼저 NameError가 발생했기 때문에, 위 코드는 Name Error이라는 문장을 출력하게 됩니다.

```
try:
    i + 1 # NameError
    1 + '1' = 0 # TypeError
except (TypeError, NameError):
    print('Type Error or Name Error')
```

위의 코드 예시는 복수 개의 에러 처리를 하나의 튜플로 묶어서 처리할 수 있도록 작성한 코드입니다. 코드 실행 시, Type Error or Name Error이라는 문장이 출력되게 됩니다.

```
try:
    i + 1
except Exception as e:
    print(e) # name 'i' is not defined
```

다음은, Exception이란 키워드인데요, Exception은 파이썬에서 예외를 다루기 위해 미리 정의된 예외 전용 클래스(class)입니다. 위 코드와 같이 except 구문 다음에 Exception as 별칭 형태로 작성하면, 예외와 연관된 값을 가질 수 있습니다.

기본 except 구문의 경우엔 프로그램이 오류에 의해 강제 종료된 후에 어떤 예외에 의해서 종료되었는지 확인할 수 있습니다.

반대로, 위 예시 코드와 같이 작성하면, 프로그램이 종료되지 않음(예외 처리)과 동시에 예외가 발생한 이유를 알 수 있습니다. (참고로, 인자의 존재와 타입은 예외에 의존적입니다.)

```
def division():
    print(1 / 0)

def main():
    try:
        division()
    except Exception as e:
        print(e)
main() # division by zero
```

머리글에서, 예외가 감지된 지점에서 직, 간접적으로 호출 한 예외 처리 코드 블록에 의해 프로그램 오류가 처리된다고 말씀드렸습니다. 앞서 살펴본 내용은 모두 직접적으로 코드 블록을 try ~ except 구문으로 묶어 처리하고 있습니다.

반대로, 위 코드 예시는 main이라는 함수에서 division이라는 1을 0으로 나누는 코드를 작성한 함수를 호출하는 경우에 예외를 처리하고 있습니다.

이처럼, 직접적으로 코드 블록을 예외 처리 구문으로 묶어주지 않더라도, 예외를 처리하고 싶은 코드 구문에서 try ~ except 구문을 작성하면, 예외를 간접적으로 처리할 수 있습니다.

finally

이번 절에서는 예외 처리 시에 함께 자주 등장하는 finally 구문에 대해서 알아보도록 하겠습니다. 결론부터 말씀드리면, finally 구문은 **예외 발생 여부와 상관없이**, try ~ except 구문이 종료되면 무조건 실행됩니다.

```
try:
     1 / 0
except Exception as e:
    print(e)
finally:
    print('run finally')

'''
코드 실행 결과:
division by zero
run finally
'''
```

위 코드를 실행하면, except 구문의 에러의 이유를 출력하는 코드 구문과 함께 finally 구문의 출력 구문이 실행되는 것을 확인할 수 있습니다. try 구문의 실행 코드가 에러 없이 작동하더라도, 마찬가지로 run finally가 출력되게 됩니다.

다음으로, 한 가지 다른 코드 예시를 보도록 하겠습니다.

```
try:
    1 / 0 # 1번 코드
except Exception as e:
    print(e) # 2번  코드
    1 / 0  # 4번 코드
finally:
    print('run finally') # 3번 코드
```

위 예시는 약간 억지스러운 코드이지만, 한 번 살펴보도록 하겠습니다.

1. 코드가 실행되면, 1번 코드에서 ZeroDivisionError: 예외가 발생합니다.
2. 예외가 발생했기 때문에, except 구문에 들어오게 됩니다. print(e) 구문에 의해 division by zero를 출력합니다.
3. 여기서 중요한데요. 예외를 출력하고, 다음 코드를 진행하는 것이 아닌, finally 구문으로 넘어가서 run finally를 출력합니다.
4. finally 구문이 종료되고, 다시 except 구문안의 1 / 0 코드를 실행하고, 예외 처리 도중에 또 다른 예외가 발생했다는 메시지와 함께 예외가 발생하며 프로그램이 종료됩니다.

결국, 아래와 같은 출력문이 터미널에 나타나게 됩니다.

```
division by zero
run finally
Traceback (most recent call last):
  File "main.py", line 19, in <module>
    1 / 0
ZeroDivisionError: division by zero

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "main.py", line 22, in <module>
    1 / 0
ZeroDivisionError: division by zero
```

일반적으로 파이썬 코드상에서 외부 자원(프로그램)을 사용하는 경우에는 에러 유무와 상관없이, 파이썬 실행 코드가 종료되는 시점에 사용한 외부자원을 반납하는 작업이 필요한 경우가 있습니다. 이러한 경우에 보통 `finally` 구문에 반납하는 코드를 작성하게 됩니다. (`e.g., 파일 읽기 또는 쓰기, 데이터베이스 읽기 또는 쓰기, etc.`)

raise

다음은, 마지막 절인 `raise` 구문에 대해서 알아보겠습니다. `raise` 구문은 강의에서도 살펴보았듯이, 개발자가 의도적으로 예외를 발생시키기 위한 구문입니다.

```
test = "Seongjae Moon"
if len(test) > 5:
    raise Exception('The text is too long.') # Exception: The text is too long.
```

위 코드 예시와 같이 특정 조건에 만족하는 경우 의도적으로 오류를 발생시키고 싶을 경우에, `raise Exception('예외 설명')` 형태로 작성할 수 있습니다.

직접 예외 명을 새로 정의할 수도 있지만, 아래와 같이 기존에 파이썬에서 제공하는 예외를 이용해서 의도적으로 예외를 발생시킬 수도 있습니다.

```
test = "Seongjae Moon"
if len(test) > 5:
    raise NameError # NameError
```

여기 까지 보셨다면, 이런 생각이 드실 수도 있습니다.

> 굳이 왜 예외를 의도적으로 발생시키지?

사실, 대부분의 경우에 가능하다면 예외 구문을 추가하는 것보다는 조건문(`if ~ elif ~ else`)을 이용해서 처리하는 것이 더 권장됩니다. 하지만, 생각보다 많은 이유에 의해서 프로그램 오류가 발생합니다.

한 가지 예를 들면, 우리가 자주 사용하는 스마트폰의 앱이 알 수 없는 이유로 오류가 발생하여 꺼지는 현상을 종종 경험하신 적이 있으실 겁니다. 이렇듯, 프로그램의 모든 오류를 개발자가 예측하고, 하나하나 조건 처리를 한다는 것은 현실적으로 많은 어려움이 따르는 작업입니다.

결론적으로, 오류가 발생하지 않으면 좋겠지만, 100% 결함이 없는 프로그램은 쉽게 구현되기 어렵습니다. 그렇기 때문에, 특정 코드 구문이 프로그램에 치명적일 수 있는 경우에는 의도적으로 예외를 발생시켜 해당 코드의 오류를 알려야하며, 오류가 발생하여 코드 실행 도중 프로그램이 강제 종료되는 일을 경험하기 전에, 적절한 대비를 통해 이를 분기할 수 있도록 하는 것이 낫다는 것이 명백합니다.

이번 강의에선 예외 처리에 대해서 알아보았습니다. 다양한 종류의 오류 종류에 대해서 알아보시고, 파이썬 코드 실행 도중에 어떤 이유 때문에 오류가 발생할 수 있는지에 대해 공부하는 것만으로도 많은 공부가 될 수 있을 거라 말씀드리고 싶습니다. :)