IPA 주관 인공지능센터 기본(fundamental) 과정

- **GitHub link: here**
- **E-Mail: windkyle7@gmail.com**

에러 (Error) 와 예외 (Exception)

문법 에러

문법 에러는 파싱(**parsing**) 에러라고도 알려져 있다. 파이썬 기본 문법에 어긋나는 코드를 작성했을 경우, `SyntaxError` 를 발생시킨다.

In [1]:

```
a = 1
for if a < 0
```

```
  File "<ipython-input-1-8879a343fc74>", line 2
    for if a < 0
        ^
SyntaxError: invalid syntax
```

예외 (Exception)

문장이나 표현식이 문법적으로 올바르다 할지라도 실행하려고 하면 에러를 일으킬 수 있다. 실행 중에 감지되는 에러들을 `예외` 라고 부르며, 무조건 치명적이지는 않다. 하지만 대부분의 예외는 프로그램이 처리하지 않아서 아래의 코드에서 볼 수 있듯이 에러 메시지를 만든다.

In [2]:

```
10 * (1 / 0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-1c40f3fcb3bb> in <module>
----> 1 10 * (1 / 0)

ZeroDivisionError: division by zero
```

In [3]:

```
4 + spam * 3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-c8ef1c0a2ff8> in <module>
----> 1 4 + spam * 3
```

```
NameError: name 'spam' is not defined
```

In [4]:

```
'2' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-d2b23a1db757> in <module>
----> 1 '2' + 2

TypeError: must be str, not int
```

파이썬에 기본적으로 `built-in` 에 내장되어 있는 내장 예외를 확인해보면 아래와 같다.

In [5]:

```
import builtins
```

In [6]:

```
errors = [error for error in dir(builtins) if 'Error' in error or
'Exception' in error]
```

In [7]:

```
errors
```

Out[7]:

```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'EOFError',
 'EnvironmentError',
 'Exception',
 'FileExistsError',
 'FileNotFoundError',
 'FloatingPointError',
 'IOError',
 'ImportError',
 'IndentationError',
 'IndexError',
 'InterruptedError',
 'IsADirectoryError',
 'KeyError',
 'LookupError',
 'MemoryError',
 'ModuleNotFoundError',
 'NameError',
 'NotADirectoryError',
```

```
 'NotImplementedError',
 'OSError',
 'OverflowError',
 'PermissionError',
 'ProcessLookupError',
 'RecursionError',
 'ReferenceError',
 'RuntimeError',
 'SyntaxError',
 'SystemError',
 'TabError',
 'TimeoutError',
 'TypeError',
 'UnboundLocalError',
 'UnicodeDecodeError',
 'UnicodeEncodeError',
 'UnicodeError',
 'UnicodeTranslateError',
 'ValueError',
 'ZeroDivisionError']
```

파이썬에서 기본적으로 제공하는 내장 예외는 총 **50**가지, 그 중 **base-class**를 제외한 나머지 **48**가지가 포함되어있다.

In [8]:

```
len(errors)
```

Out[8]:

```
50
```

파이썬에서는 모든 예외가 `BaseException`에서 파생된 클래스의 인스턴스여야 한다. 특정 클래스를 언급하는 `except` 절을 갖는 `try` 문에서, 그 절은 그 클래스에서 파생된 모든 예외 클래스를 처리한다. **(**하지만 이를 상속받는 예외 클래스는 처리하지 않는다. 또한 상속을 통해 관련되지 않은 두 개의 예외 클래스는 같은 이름을 갖는다 할지라도 결코 등등하게 취급되지 않는다**.)**

예외 처리: (try-except-else 문)

파이썬에서는 `EAFP(Easier to ask for forgiveness than permission)` 라는 철학을 가지고 있어 흔히 볼 수 있는 파이썬 코딩 스타일은 올바른 **Key**나 **Attribute**의 존재를 가정하고, 그 가정이 틀리면 예외를 잡는다. 이 스타일은 많은 `try`와 `except` 문의 존재로 특징지어진다. 이 기법은 **C** 언어와 같은 다른 많은 언어에서 자주 사용되는 `LBYL` 스타일과 대비된다.

아래의 코드는 문자열에 정수형을 연산하려고 하므로 `TypeError` 예외를 발생시켜본다.

In [9]:

```
a = '1'
b = a + 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-d7009c2014a9> in <module>
      1 a = '1'
```

```
----> 2 b = a + 1

TypeError: must be str, not int
```

`try-except` 문을 사용하여 예외가 발생하면 `except` 문을 수행하도록 한다.

In [10]:

```
try:
    a = '1'
    b = a + 1
except:
    print('Error')
else:
    print('result:', b)
```

```
Error
```

`else` 문은 예외가 발생하지 않고 완벽하게 수행했을 경우 수행된다.

In [11]:

```
try:
    a = 1
    b = a + 1
except:
    print('Error')
else:
    print('result:', b)
```

```
result: 2
```

assert

`assert` 는 결과가 `거짓(False)` 일 경우 스코프가 없다면 `AssertionError` 예외를 발생시킨다.

In [12]:

```
assert False
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-12-a871fdc9ebee> in <module>
----> 1 assert False

AssertionError:
```

In [13]:

```
assert True
```

아래의 코드처럼 예외가 발생했을 때 해당 예외에 대한 메시지를 출력하도록 할 수 있다.

In [14]:

```
a = 1
```

```
a = 1
solution = '''a가 1보다 작아야 합니다.
ex) a = 0
'''
assert a < 1, solution
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-14-5c8cd0741605> in <module>
      3 ex) a = 0
      4 '''
----> 5 assert a < 1, solution

AssertionError: a가 1보다 작아야 합니다.
ex) a = 0
```

raise

`raise`는 에러를 강제로 발생하도록 하는 키워드이다.

In [15]:

```
def f(x):
    if x > 0:
        raise
    elif x < 0:
        raise KeyError
```

위의 함수 `f`에서 **x**가 **0**보다 큰 경우에는 `raise`로 인해 예외가 발생한다. 현재 스코프에 활성화된 예외가 없다면, 이것이 에러라는 것을 알리기 위해 `RuntimeError` 예외를 일으킨다.

In [16]:

```
f(1)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-16-281ab0a37d7d> in <module>
----> 1 f(1)

<ipython-input-15-cbd659995a01> in f(x)
      1 def f(x):
      2     if x > 0:
----> 3         raise
      4     elif x < 0:
      5         raise KeyError

RuntimeError: No active exception to reraise
```

x가 **0**보다 작은 경우, 현재 스코프에 정의된 `KeyError` 예외를 발생시킨다.

In [17]:

```
f(-1)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

```
<ipython-input-17-8e6746a6b70d> in <module>
----> 1 f(-1)

<ipython-input-15-cbd659995a01> in f(x)
      3         raise
      4     elif x < 0:
----> 5         raise KeyError

KeyError:
```