Exceptions

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Python

Exceptions & Assertions

- Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them.
  - Exception Handling
  - Assertions

Standard Exceptions

- **`Exception`**
  - Base class for all exceptions
- **`StopIteration`**
  - Raised when the **`next()`** method of an iterator does not point to any object.
- **`SystemExit`**
  - Raised by the **`sys.exit()`** function.
- **`StandardError`**
  - Base class for all built-in exceptions except **`StopIteration`** and **`SystemExit`**.

Standard Exceptions (Cont.)

- **`ArithmeticError`**
  - Base class for all errors that occur for numeric calculation.
- **`OverflowError`**
  - Raised when a calculation exceeds maximum limit for a numeric type.
- **`FloatingPointError`**
  - Raised when a floating point calculation fails.
- **`ZeroDivisonError`**
  - Raised when division or modulo by zero takes place for all numeric types.

Standard Exceptions (Cont.)

- **`AssertionError`**
  - Raised in case of failure of the Assert statement.
- **`AttributeError`**
  - Raised in case of failure of attribute reference or assignment.
- **`EOFError`**
  - Raised when there is no input from either the **`raw_input()`** or **`input()`** function and the end of file is reached.
- **`ImportError`**
  - Raised when an import statement fails.

Standard Exceptions (Cont.)

- **`KeyboardInterrupt`**
  - Raised when the user interrupts program execution, usually by pressing `Ctrl+c`.
- **`LookupError`**
  - Base class for all lookup errors.
- **`IndexError`**
  - Raised when an index is not found in a sequence.
- **`KeyError`**
  - Raised when the specified key is not found in the dictionary.

Standard Exceptions (Cont.)

- **`NameError`**
  - Raised when an identifier is not found in the local or global namespace.
- **`UnboundLocalError`**
  - Raised when trying to access a local variable in a function or method but no value has been assigned to it.
- **`EnvironmentError`**
  - Base class for all exceptions that occur outside the Python environment.

Standard Exceptions (Cont.)

- **`IOError`**
  - Raised when an input/ output operation fails, such as the print statement or the `open()` function when trying to open a file that does not exist.
- **`OSError`**
  - Raised for operating system-related errors.
- **`SyntaxError`**
  - Raised when there is an error in Python syntax.
- **`IndentationError`**
  - Raised when indentation is not specified properly.

Standard Exceptions (Cont.)

- **`SystemError`**
  - Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
- **`SystemExit`**
  - Raised when Python interpreter is quit by using the `sys.exit()` function. If not handled in the code, causes the interpreter to exit.
- **`TypeError`**
  - Raised when an operation or function is attempted that is invalid for the specified data type.

Standard Exceptions (Cont.)

- **`ValueError`**
  - Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
- **`RuntimeError`**
  - Raised when a generated error does not fall into any category.
- **`NotImplementedError`**
  - Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

What is Exception?

- Is an ***event***.
- Occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
- An exception is a Python object that represents an error.

What is Exception? (Cont.)

- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

- If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a `try:` block.
- After the `try:` block, include an `except:` statement, followed by a block of code which handles the problem as elegantly as possible.

Handling an exception (Cont.)

- Syntax

```
try:
    You do your operations here
    ......................
except ExceptionI:
    If there is ExceptionI, then execute this block.
except ExceptionII:
    If there is ExceptionII, then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

Handling an exception (Cont.)

- A single `try` statement can have multiple `except` statements.
- This is useful when the `try` block contains statements that may throw different types of exceptions.
- Can also provide a generic `except` clause, which handles any exception.

Handling an exception (Cont.)

- After the **`except`** clause(s), can include an **`else-clause`**.
- The code in the **`else-block`** executes if the code in the **`try:`** block does not raise an exception.
- The **`else-block`** is a good place for code that does not need the **`try:`** block's protection.

Handling an exception (Cont.)

```
1 try:
2     fh = open("testfile", "w")
3     fh.write("This is my test file for exception handling!!")
4 except IOError:
5     print ("Error: can\'t find file or read data")
6 else:
7     print ("Written content in the file successfully")
8     fh.close()
9
```

Console Problems PyUnit

<terminated> demo.py [/usr/bin/python3.6]

```
Written content in the file successfully
```

Handling an exception (Cont.)

```
1 try:
2     fh = open("testfile", "r")
3     fh.write("This is my test file for exception handling!!")
4 except IOError:
5     print ("Error: can\'t find file or read data")
6 else:
7     print ("Written content in the file successfully")
8
9
```

Console | Problems | PyUnit

<terminated> demo.py [/usr/bin/python3.6]

```
Error: can't find file or read data
```

The except Clause with No Exceptions

- Can also use the except statement with no exceptions defined as follows :

```
try:
    You do your operations here
    ......................
except:
    If there is any exception, then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

The except Clause with No Exceptions (Cont.)

- This kind of a `try-except` statement catches all the exceptions that occur.
- Using this kind of `try-except` statement is not considered a good programming practice though, because it catches ***all*** exceptions but does not make the programmer identify the root cause of the problem that may occur.

The except Clause with Multiple Exceptions

- You can also use the same except statement to handle multiple exceptions as follows :

```
try:
   You do your operations here
   ......................
except(Exception1[, Exception2[,...ExceptionN]]):
   If there is any exception from the given exception
    list, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

The try-finally Clause

- Can use a `finally:` block along with a `try:` block.
- The `finally:` block is a place to put any code that ***must*** execute, whether the try-block raised an exception or not.

The try-finally Clause (Cont.)

```
try:
    You do your operations here;
    ......................
    Due to any exception, this may be skipped.
finally:
    This would always be executed.
    ......................
```

The try-finally Clause (Cont.)

```
1 try:
2     fh = open("testfile", "w")
3     fh.write("This is my test file for exception handling!!")
4 finally:
5     print ("Error: can\'t find file or read data")
6     fh.close()
7
8
9
```

Console | Problems | PyUnit

<terminated> demo.py [/usr/bin/python3.6]

```
Error: can't find file or read data
```

The try-finally Clause (Cont.)

```
1  try:
2      fh = open("testfile", "w")
3      try:
4          fh.write("This is my test file for exception handling!!")
5      finally:
6          print ("Going to close the file")
7          fh.close()
8  except IOError:
9      print ("Error: can\'t find file or read data")
10
11
```

Console Problems PyUnit

<terminated> demo.py [/usr/bin/python3.6]

```
Going to close the file
```

Argument of an Exception

- An exception can have an argument, which is a value that gives additional information about the problem.
- The contents of the argument vary by exception.

```
try:
   You do your operations here
   ......................
except ExceptionType as Argument:
   You can print value of Argument here...
```

Argument of an Exception (Cont.)

```
1  # Define a function here.
2
3  def temp_convert(var):
4      try:
5          return int(var)
6      except ValueError as Argument:
7          print ("The argument does not contain numbers\n", Argument)
8
9  # Call above function here.
10 temp_convert("xyz")
11
```

Console Problems PyUnit

<terminated> demo.py [/usr/bin/python3.6]

```
The argument does not contain numbers
 invalid literal for int() with base 10: 'xyz'
```

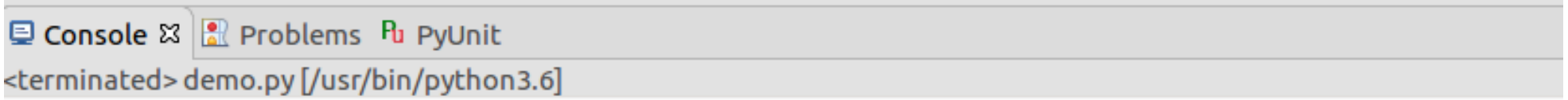
Raising an Exception

- Can raise exceptions in several ways by using the **`raise`** statement.
- Syntax

```
raise [Exception [, args [, traceback]]]
```

Raising an Exception (Cont.)

```
1  def functionName( level ):
2      if level <1:
3          raise Exception(level)
4          # The code below to this would not be executed
5          # if we raise the exception
6      return level
7
8  try:
9      l = functionName(-10)
10     print ("level = ",l)
11 except Exception as e:
12     print ("error in level argument",e.args[0])
13
```




```
error in level argument -10
```

User-Defined Exceptions

- Python also allows to create your own exceptions by deriving classes from the standard built-in exceptions.

```
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg
```

User-Defined Exceptions (Cont.)

- Here is an example related to `RuntimeError`.
- Here, a class is created that is subclassed from `RuntimeError`.
- This is useful when you need to display more specific information when an exception is caught.
- In the `try` block, the user-defined exception is raised and caught in the except block.
- The variable `e` is used to create an instance of the class `Networkerror`.

User-Defined Exceptions (Cont.)

- So once you have defined the above class, you can raise the exception as follows :

```
try:
    raise Networkerror("Bad hostname")
except Networkerror,e:
    print e.args
```

Assertions in Python

- Is a sanity-check that you can turn on or turn off when you are done with your testing of the program.
- The easiest way to think of an assertion is to liken it to a **`raise-if`** statement (or to be more accurate, a **`raise-if-not statement`**).
- An expression is tested, and if the result comes up **`false`**, an exception is raised.

Assertions in Python (Cont.)

- Assertions are carried out by the `assert` statement, the newest keyword to Python, introduced in version 1.5.
- Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

The assert Statement

- When it encounters an **`assert`** statement, Python evaluates the accompanying expression, which is hopefully true.
- If the expression is **`false`**, Python raises an **`AssertionError`** exception.
- Syntax

```
assert Expression[, Arguments]
```

The assert Statement (Cont.)

- If the assertion **`fails`**, Python uses **`ArgumentExpression`** as the argument for the **`AssertionError`**.
- **`AssertionError`** exceptions can be caught and handled like any other exception, using the **`try-except`** statement.
- If they are not handled, they will terminate the program and produce a traceback.

The assert Statement (Cont.)

```
1 def KelvinToFahrenheit(Temperature):
2     assert (Temperature >= 0),"Colder than absolute zero!"
3     return ((Temperature - 273) * 1.8) + 32
4
5 print (KelvinToFahrenheit(273))
6 print (int(KelvinToFahrenheit(505.78)))
7 print (KelvinToFahrenheit(-5))
8
```

Console Problems PyUnit

<terminated> demo.py [/usr/bin/python3.6]

```
32.0
451
Traceback (most recent call last):
  File "/home/instructor/PythonHome/0831/demo.py", line 7, in <module>
    print (KelvinToFahrenheit(-5))
  File "/home/instructor/PythonHome/0831/demo.py", line 2, in KelvinToFahrenheit
    assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

Lab

```
1 try:
2     print('안녕하세요.')
3     print(param)
4 except:
5     print('예외가 발생했습니다!')
6
```

Console | Problems | PyUnit

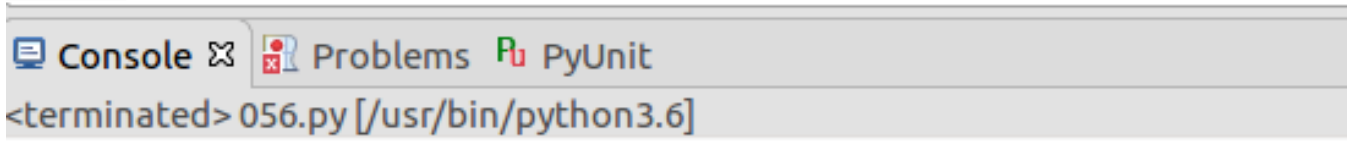
<terminated> 055.py [/usr/bin/python3.6]

안녕하세요.

예외가 발생했습니다!

Lab (Cont.)

```
1 try:
2     print('안녕하세요.')
3     print(param)
4 except:
5     print('예외가 발생했습니다!')
6 else:
7     print('예외가 발생하지 않았습니다.')
8
9
```

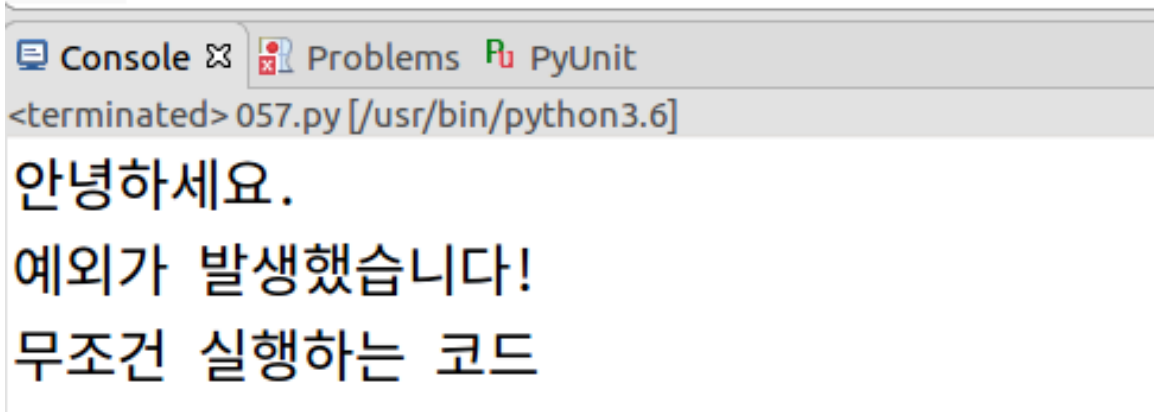


안녕하세요.

예외가 발생했습니다!

Lab (Cont.)

```
1 try:
2     print('안녕하세요.')
3     print(param)
4 except:
5     print('예외가 발생했습니다!')
6 finally:
7     print('무조건 실행하는 코드')
8
9
```



Lab (Cont.)

```
1 try:
2     print(param)
3 except Exception as e:
4     print(e)          # name 'param' is not defined 가 출력됨
5
```

Console Problems PyUnit

<terminated> 058.py [/usr/bin/python3.6]

```
name 'param' is not defined
```

Lab (Cont.)

```
1 import time
2 count = 1
3 try:
4     while True:
5         print(count)
6         count += 1
7         time.sleep(0.5)
8 except KeyboardInterrupt:   # Ctrl-C가 입력되면 발생하는 오류
9     print('사용자에 의해 프로그램이 중단되었습니다.')
10
```