Exception

Exception (มาจากคำเต็มๆว่า Exceptional Event)คือ เหตุการณ์ที่เกิดขึ้นขณะตัวโปรแกรมกำลังรันอยู่แล้วเกิดข้อผิดพลาดขึ้นจนทำให้โปรแกรมไม่ทำงานหรือเกิด error ภายในโปรแกรมนั้น เมื่อใดก็ตามที่เกิด runtime error ขึ้น โปรแกรมจะสร้าง exception โดยอัตโนมัติ โดยปกติแล้วโปรแกรมจะหยุดทำงาน และ Python จะพิมพ์ข้อความแสดงข้อผิดพลาดให้ ซึ่งข้อความที่แสดงข้อผิดพลาดจะมีอยู่ 2 ส่วน คือ ชนิดของความผิดพลาดจะอยู่ข้างหน้าเครื่องหมาย : และส่วนขยายความหลังเครื่องหมาย

เช่น

ZeroDivisionError: integer division or modulo

(การเข้าไปใช้งาน ในส่วนที่ไม่มีอยู่จริง)

IndexError: list index out of range

(การเข้าไปที่ key ที่มันไม่มีอยู่ในdirectory)

Exception มาตรฐานที่มีอยู่ใน Python

ชื่อ Exception	รายละเอียด
Exception	คลาสพื้นฐานของข้อผิดพลาดทั้งหมด
StopIteration	เป็นข้อผิดพลาดที่เกิดเมื่อเมทธอดถัดไปของการทำซ้ำไม่ได้ชี้ไปยังวัตถุใดๆ
SystemExit	เป็นข้อผิดพลาดที่เกิดเมื่อโดยฟังก์ชันsys.exit()
StandardError	คลาสพื้นฐานของexceptionsทั้งหมดที่เกิดขึ้นยกเว้น StopIteration และ SystemExit
ArithmeticError	คลาสพื้นฐานของข้อผิดพลาดทั้งหมดที่เกิดขึ้นในการคำนวณตัวเลข
OverflowError	เป็นข้อผิดพลาดที่เกิดเมื่อคำนวณเกินค่าสูงสุดของชนิดข้อมูลที่เป็นตัวเลข
FloatingPointError	เป็นข้อผิดพลาดที่เกิดเมื่อคำนวณ floating point ล้มเหลว
ZeroDivisonError	เป็นข้อผิดพลาดที่เกิดเมื่อ Divisionหรือ

	modulo ด้วยศูนย์จะเกิดขึ้นกับชนิดข้อมูลที่เป็นตัวเลขทั้งหมด
AssertionError	เป็นข้อผิดพลาดที่เกิดในกรณีที่ Assert statement เกิดข้อผิดพลาด
AttributeError	เป็นข้อผิดพลาดที่เกิดในกรณีที่การอ้างอิงหรือการกำหนดแอตทริบิวต์ผิด
EOFError	เป็นข้อผิดพลาดที่เกิดเมื่อมีการป้อนข้อมูลทั้งที่ไม่มีฟังก์ชัน raw_input() หรือinput() และระบุจุดสิ้นสุดของไฟล์ไม่ถูกต้อง
ImportError	เป็นข้อผิดพลาดที่เกิดเมื่อ import statement ผิด
KeyboardInterrupt	เป็นข้อผิดพลาดที่เกิดเมื่อผู้ใช้ขัดจังหวะการทำงานของโปรแกรมโดยมักจะกด Ctrl + C
LookupError	คลาสพื้นฐานของข้อผิดพลาดทั้งหมดของการค้นหา
IndexError	เป็นข้อผิดพลาดที่เกิดเมื่อไม่พบ index ในลำดับ
KeyError	เป็นข้อผิดพลาดที่เกิดเมื่อไม่พบคีย์ที่ระบุในดิกชันนารี
NameError	เป็นข้อผิดพลาดที่เกิดเมื่อไม่พบ identifier ใน local หรือ global namespace
UnboundLocalError	เป็นข้อผิดพลาดที่เกิดเมื่อพยายามเข้าถึง local variable ในฟังก์ชันหรือเมทธอด แต่ยังไม่มีการกำหนดค่าให้มัน
EnvironmentError	คลาสพื้นฐานของข้อผิดพลาดทั้งหมดที่เกิดขึ้นภายนอกสภาพแวดล้อมของ Python
IOError	เป็นข้อผิดพลาดที่เกิดจากตัวดำเนินการในการ input/ output เช่นคำสั่ง print หรือฟังก์ชัน open() เมื่อพยายามที่จะเปิดไฟล์ที่ไม่มีอยู่
OSError	เป็นข้อผิดพลาดในการดำเนินงาน systemrelated
SyntaxError	เป็นข้อผิดพลาดที่เกิดจากการผิดไวยากรณ์ของ Python

IndentationError	เป็นข้อผิดพลาดที่เกิดเมื่อindentationไม่ได้ระบุอย่างถูกต้อง
SystemError	เป็นข้อผิดพลาดที่เกิดเมื่อ Interpreter พบว่าเป็นข้อผิดพลาดภายใน แต่เมื่อพบข้อผิดพลาดนี้ Interpreter ไม่จบการทำงาน
SystemExit	เป็นข้อผิดพลาดที่เกิดเมื่อ Interpreter จะจบการทำงานโดยใช้ฟังก์ชัน sys.exit () หากไม่ได้จัดการกับโค้ด ทำให้ Interpreter จบการทำงาน
TypeError	เป็นข้อผิดพลาดที่เกิดเมื่อ operation หรือฟังก์ชัน ระบุชนิดข้อมูลที่ไม่ถูกต้อง
ValueError	เป็นข้อผิดพลาดที่เกิดเมื่อสร้างฟังก์ชั่นที่มีชนิดข้อมูลเดียวกับอาร์กิวเมนต์ แต่ อาร์กิวเมนต์มีค่าที่ไม่ถูกต้อง
RuntimeError	เป็นข้อผิดพลาดที่เกิดเมื่อข้อผิดพลาดที่เกิดขึ้นไม่ได้จัดอยู่ในประเภทใดๆ
NotImplementedError	เป็นข้อผิดพลาดที่เกิดเมื่อ abstract method ที่จะต้องมีการ implemented ใน inherited class ไม่ได้นำมาใช้จริง

Exception Handling :

บางครั้งโปรแกรมต้องดำเนินการ execute ที่เป็นเหตุของ exception (ข้อยกเว้น) แต่ไม่ต้องการให้โปรแกรมหยุด เราสามารถจัดการกับข้อยกเว้น โดยการใช้ try และ except statement การดักจับข้อมูลใน Python ประกอบไปด้วย try...except , try...finally, raise

รูปแบบ การใช้งาน **try...except**

```
try:
   statement(s)
except [expression [, target]]:
   statement(s)
[else: statement(s)]
```

เช่น

```
1/0 #
```

ผลการรันจะได้

```
ZeroDivisionError: division by zero
```

หากใช้ try...exept จะสามารถแจ้งข้อมูลเป็นอย่างอื่นให้ผู้ใช้ได้อ่านได้

ตัวอย่าง

```
try:
   1/0
except :
   print "ไม่สามารถ หารด้วย 0 ได้"
```

ผลที่ได้จากการรัน

```
ไม่สามารถหารด้วย 0 ได้
```

สามารถระบุข้อหลายผิดพลาดในครั้งเดียว

```
except (RuntimeError, TypeError, NameError):
        ……..
```

หรือดักจับข้อผิดพลาดเป็นส่วนๆ

```
try:
   f = open('myfile.txt')
   s = f.readline()
   i = int(s.strip())
except IOError, (errno, strerror):
   print "I/O error(%s): %s" % (errno, strerror)
except ValueError:
   print "Could not convert data to an integer."
except:
   print "Unexpected error:", sys.exc_info()[0]
   raise
```

สามารถใช้ else: ในกรณีที่ try: ไม่ยอมแจ้งข้อผิดพลาด

รูปแบบการใช้งาน ***try....except...else***

```
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.
```

ตัวอย่าง

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

Raising Exceptions :

รูปแบบการใช้งาน **raise**

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

สามารถยกข้อผิดพลาดมาแสดงได้ เช่น

```
raise NameError, 'HiThere'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: HiThere
```

พารามิเตอร์ตัวแรกเป็นชื่อข้อผิดพลาด ตัวหลังเป็นข้อความการผิดพลาด ซึ่งอาจเขียนอีกรูปหนึ่งว่า raise NameError('HiThere')

สามารถใช้คำสั่ง raise เพื่อยกข้อผิดพลาดขึ้นอีกครั้ง ภายในบล็อกที่ดักความผิดพลาดได้

```
try:
 raise NameError, 'HiThere'
 except NameError:
 print 'An exception flew by!'
 raise
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: HiThere
```

User-defined Exceptions :

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

สามารถสร้าง Exception เองโดยสร้างเป็นคลาส ซึ่งเวลานำมาใช้ต้องเรียกมาจากคลาสที่สร้างไว้อีกที

```
class MyError(Exception):
        def __init__(self, value):
        self.value = value
        def __str__(self):
        return repr(self.value)
...
 try:
        raise MyError(2*2)
        except MyError, e:
        print 'My exception occurred, value:', e.value
...
My exception occurred, value: 4

raise MyError, 'oops!'
```

```
Traceback (most recent call last):
 File "<stdin>", line 1, in ?
__main__.MyError: 'oops!'
```

จากตัวอย่างนี้ เมธอด __init__ ของคลาสเริ่มคือ Exception ถูกครอบด้วยโค้ด

หากมอดูลต้องมีการดักข้อผิดพลาดหลายอย่าง ควรสร้างคลาสที่ใช้เป็นฐานก่อน แล้วจึงเรียกใช้จากคลาสฐานมาอีกทีหนึ่ง เวลาเปลี่ยนแปลงอะไรจะทำได้ง่ายกว่า คือทำที่คลาสฐานที่เดียว

```
class Error(Exception):
   """Base class for exceptions in this module."""
   pass

class InputError(Error):
   """Exception raised for errors in the input.

   Attributes:
      expression -- input expression in which the error occurred
      message -- explanation of the error
   """

   def __init__(self, expression, message):
      self.expression = expression
      self.message = message

class TransitionError(Error):
   """Raised when an operation attempts a state transition that's
not
   allowed.

   Attributes:
      previous -- state at beginning of transition
      next -- attempted new state
      message -- explanation of why the specific transition is not
allowed
```

```
    """

    def __init__(self, previous, next, message):
        self.previous = previous
        self.next = next
        self.message = message
```

Defining Clean-up Actions :

ในประโยค try : จะมีวลีเผื่อเลือกอีกตัวคือ finally : โค้ดที่อยู่ในส่วนนี้จะถูกรันเสมอ ไม่ว่าจะเกิดข้อผิดพลาดขึ้นหรือไม่

รูปแบบการใช้งาน **try...finally**

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

ตัวอย่าง

```
try:
...     raise KeyboardInterrupt
finally:
...     print 'Goodbye, world!'
...
Goodbye, world!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
KeyboardInterrupt
```

ประโยชน์ของ finally : คือ ถ้าผ่านบล็อกของ except : และ else : มาแล้ว ถ้ายังไม่แจ้งข้อผิดพลาด จะถูกแจ้งในส่วนนี้ และไม่ว่าจะพบคำสั่ง break continue หรือ return ก็ตาม โค้ดในส่วนนี้จะถูกรันเสมอ

```
def divide(x, y):
try:
        result = x / y
        except ZeroDivisionError:
        print "division by zero!"
else:
        print "result is", result
         finally:
         print "executing finally clause"
...
divide(2, 1)
        result is 2
        executing finally clause
divide(2, 0)
        division by zero!
        executing finally clause

divide("2", "1")
        executing finally clause
        Traceback (most recent call last):
        File "<stdin>", line 1, in ?
        File "<stdin>", line 3, in divide
        TypeError: unsupported operand type(s) for /: 'str' and
'str'
```

Predefined Clean-up Actions :

ควรมีความรอบคอบในการเขียนโค้ด

```
for line in open("myfile.txt"):
   print line
```

ปัญหาของโค้ดนี้คือ ไม่มีการปิดไฟล์ ถ้าโปรแกรมใหญ่ขึ้นโค้ดแบบนี้จะใช้พื้นที่ในหน่วยความจำมาก ดังนั้นโค้ดควรเป็นดังนี้

```
with open("myfile.txt") as f:
   for line in f:
      print line
```

ตัวอย่างนี้ ไฟล์ออปเจคต์ f จะถูกลบออกจากหน่วยความจำเมื่อโปรแกรมจบ