Exception Handling

BCA531 - Python Programming

Dr. M. Prabu
Dr. M. Umme Salma

Contents

- Exceptions
- Throwing and catching exceptions
- try except
- try except else
- try except finally
- Common python exceptions

What is an Exception?

- An exception is an error that happens during execution of a program
- If an exception is not caught the program is terminated
- In Python, exceptions are triggered automatically on errors, and they can be triggered and intercepted by your code

Exceptions

- Exception handling has two steps:
  - Raising or Throwing
  - Catching

- Python provides 3 keywords to deal with exceptions :
  - raise
  - try
  - except

Common Python Exceptions

Exception	Description
IOError	If the file cannot be opened
ImportError	If python cannot find the module
ValueError	Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value
KeyError	Raised when a mapping (dictionary) key is not found in the set of existing keys
IndentationError	raised due to incorrect indentation
SyntaxError	Raised when the parser encounters a syntax error

Exceptions (2)

Program to divide a constant by a number

```
def divide(num):
      print 100/num

if __name__ == '__main__':
    divide(0)
```

OUPUT:

ZeroDivisionError: **integer division or modulo by zero**

Exception propagation


```
>>> def f3(num):
    constant = 100
    return constant/num

>>> def f2(num):
    return f3(num)

>>> def f1(num):
    return f2(num)

>>> f1(10)
10
>>> f1(0)
```

```
>>> f1(0)

Traceback (most recent call last):
  File "<pyshell#10>", line 1, in
<module>
    f1(0)
  File "<pyshell#8>", line 2, in f1
    return f2(num)
  File "<pyshell#5>", line 2, in f2
    return f3(num)
  File "<pyshell#2>", line 3, in f3
    return constant/num
ZeroDivisionError: integer division
or modulo by zero
```

Why use exceptions

- Error handling: Python raises an exception whenever it detects errors in program at runtime. You can catch and respond to errors in the code or Python's default behavior kicks in, stops the program and prints the error message.

- Event notification: exceptions can also be used to signal valid conditions without you having to pass result flags around a program

Throwing and Catching Exceptions

Throwing an exception


```
def avg(seq):
    result = 0
    for val in seq:
        result += convert(val)
    return result/len(seq)

def convert(val):
    try:
        val = int(val)
    except ValueError:
        raise ValueError('val type is not int')
    return val

print avg([1, 2, 4, 5])
Output:
3
```

Throwing an exception


```
print avg([1, 'two', 4, 5])

Output:
Traceback (most recent call last):
  File "exceptions1.py", line 15, in <module>
    print avg([1, 'two', 4, 5])
  File "exceptions1.py", line 4, in avg
    result += convert(val)
  File "exceptions1.py", line 11, in convert
    raise ValueError('val type is not int')
ValueError: val type is not int
```

Handling Exceptions (try except block)

- **In order to handle exceptions wrap the code in `try except`**

```
def divide(num):
    try:
        print 100/num
    except ZeroDivisionError:
        print("division by Zero not allowed")

if __name__=='__main__':
    divide(0)
```

Output:

```
division by Zero not allowed
```

try except else


```
try:
    # do something
except:
    # handle exception
else:
    # executed only when there is no exception
```

The code in `else` block is only executed if there is no exception

try except else (2)


```
def divide(num):
    try:
        result = 100/num
    except ZeroDivisionError:
        print('division by Zero not allowed')
    else:
        print "Result is %d" % (result)

if __name__ == '__main__':
    divide(10)
```

Output:

```
Result is 10
```

try except finally

```
try:
    # do something
except:
    # handle exception
finally:
    # always executed
```

The code in `finally` block is always executed, no matter what

try except finally (2)


```
def divide(num):
    try:
        result = 100/num
    except ZeroDivisionError:
        print('division by Zero not allowed')
    finally:
        print "Input was %d" % (num)

if __name__ == '__main__':
    divide(0)
```

Output:

```
Division by Zero not allowed
Your input was 0
```

Custom exceptions


```
class MyException(Exception):
      pass

def divide(num):
      try:
            return 100/num
      except ZeroDivisionError:
            raise MyException('Cannot divide
by 0')
```