CH5: Python Data Type

Everything is all right until there is no error in program. But in real time scenario, nothing is bug free. Your program might got stoped while opening a non-existing file or throw an error while doing some invalid arithmetic operation in runtime.

Hence handling exception is really important while writing a program. There is a pythoic way to handle a runtime exception.

What is Exception? An exception is an event, which 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 can't cope with, it raises an exception. An exception is a Python object that represents an error.

99.1 Sample Example of Exception / Error

Let’s try with following example

>>> 5/0

Traceback (most recent call last):

File "<stdin>", line 1, in ?

ZeroDivisionError: integer division or modulo by zero

>>> python\*10

Traceback (most recent call last):

File "<stdin>", line 1, in ?

NameError: name python is not defined

>>> 3 + ‘python’

Traceback (most recent call last):

File "<stdin>", line 1, in ?

TypeError: cannot concatenate 'str' and 'int' objects

Let’s see how an Exception is handled by python program. Do you think the following program is bug-free ?

while True:

x = int(raw\_input("Please enter a number: "))

print 'Cool ! You Entered '+str(x) +'.'

## Yes It’s not! A sample output is:

>>>

Please enter a number: 2

Cool ! You Entered 2.

Please enter a number: 5

Cool ! You Entered 5.

Please enter a number: x

Traceback (most recent call last):

File "C:\Users\dipankar.dutta\Desktop\MyExp\python\Code\Exceptions.py", line 8, in <module>

x = int(raw\_input("Please enter a number: "))

ValueError: **invalid** literal for int() with base 10: 'x'

As you see the script got crashed while input is other than iinteger.

Python has a beautiful way to handle this station using try: except block. If we have some suspicious code that might cause any exception, we can put that suspicious code in a try block . We also add some statement in except block, handling the situation if exception occurs. Hence the correct version of the above code is below.

while True:

try:

x = int(raw\_input("Please enter a number: "))

print 'Cool ! You Entered '+str(x) +'.'

except:

print 'OOPs ! you enter other than integer'

**OUTPUT:**

Please enter a number: 1

Cool ! You Entered 1.

Please enter a number: 2

Cool ! You Entered 2.

Please enter a number: r

OOPs ! you enter other than integer

Please enter a number:

## Suppose we want to do something like: if there is no exception we will stop the game, but if there an exception we will ask again for next number. So there is need a block which do something when there is no exception. It can be achieved by adding an ELSE block after except. Thus the modified code is.

while True:

try:

x = int(raw\_input("Please enter a number: "))

print 'Cool ! You Entered '+str(x) +'.'

except:

print 'OOPs ! you enter other than integer'

else:

print 'Done! Exiting'

break;

Please enter a number: t

OOPs ! you enter other than integer

Please enter a number: p

OOPs ! you enter other than integer

Please enter a number: 6

Cool ! You Entered 6.

Done! Exiting

>>>

## There are some situations where the suspicious code might have more than one exception type and we want to show different message for different exception type. In that case, we can have multiple except block for each possible exception.

## For example, we have a file “intFile.txt” should contains an integer and we want to read that integer, display it. There might be two problems:

## File might not be there.

## File is there but File does not contain any integer.

## We can handle this situation using following code segment:

import sys

try:

f = open('myfile.txt')

s = f.readline()

i = int(s.strip())

except IOError as e:

print "I/O error({0}): {1}".format(e.errno, e.strerror)

except ValueError:

print "The File Should be an Integer"

except Exception,e:

print "Unexpected error:", e

else:

f.close()

## There are some situations, when we want do some operation regardless an exception or not. We can place this segment of codes in a finally block.The finally block is a placeholder to put any code that must execute, whether the try-block raised an exception or not. For example, we want to say “Thank You” to the user, who runs the above script.

import sys

try:

f = open('myfile.txt')

s = f.readline()

i = int(s.strip())

except IOError as e:

print "I/O error({0}): {1}".format(e.errno, e.strerror)

except ValueError:

print "The File Should be an Integer"

except Exception,e:

print "Unexpected error:", e

else:

f.close()

finally:

print 'ThankYou'

This code always print “Thank You” at end all the time.

Until this, we discussed about the python build-in Exception Handler. User can define his own exception handler.

We have a Network application which test host reachability. We design a Host Unreachable exception class to raise the exception when the host is not reachable.

class HostUnreachable(Exception):

def \_\_init\_\_(self):

self.value = 505

def \_\_str\_\_(self):

return 'Host is not reachable. ErrorCode:'+str(self.value)

try:

raise HostUnreachable()

except HostUnreachable as e:

print e