Exception handling in Python

What are exceptions in Python?

An exception is a warning or an error condition.

Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.

When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.

For example, if function A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.

If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

Catching Exceptions in Python

In Python, exceptions can be handled using a try statement.

A critical operation which can raise exception is placed inside the **try** clause and the code that handles exception is written in **except** clause.

It is up to us, what operations we perform once we have caught the exception. Here is a simple example.

Example 1(Excep 1.py)

```
# import module sys to get the type of exception

import sys

randomList = ['a', 0, 2]

for entry in randomList:

    try:

        print("The entry is", entry)

        r = 1/int(entry)

        break

    except:

        print("Oops!",sys.exc_info()[0],"occurred.")

        print("Next entry.")

        print()
print("The reciprocal of",entry,"is",r)
```

Output is:

```
The entry is a
Oops! <class 'ValueError'>
occurred.Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError' >
occurred.Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

In this program, we loop until the user enters an integer that has a valid reciprocal. The portion that can cause exception is placed inside try block.

If no exception occurs, except block is skipped and normal flow continues. But if any exception occurs, it is caught by the except block.

Here, we print the name of the exception using ex_info() function inside sys module and ask the user to try again. We can see that the values 'a' causes ValueError and '0' causes ZeroDivisionError.

Catching Specific Exceptions in Python

In the above example, we did not mention any exception in the except clause.

This is not a good programming practice as it will catch all exceptions and handle every case in the same way. We can specify which exceptions, an except clause will catch.

A try clause can have any number of except clause to handle them differently but only one will be executed in case an exception occurs.

We can use a tuple of values to specify multiple exceptions in an except clause. Here is an example pseudo code.

Example 5(Excep 5.py)

```
try:
   # do something
   pass

except ValueError:
   # handle ValueError exception
   pass

except (TypeError, ZeroDivisionError):
   # handle multiple exceptions
   # TypeError and ZeroDivisionError
   pass
```

```
except:
   # handle all other exceptions
   pass
```

Raising Exceptions

In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword **raise.**

We can also optionally pass in value to the exception to clarify why that exception was raised.

Example 2(Excep 2.py)

```
try:
...     a = int(input("Enter a positive integer: "))
...     if a <= 0:
...         raise ValueError("That is not a positive number!")
... except ValueError as ve:
...     print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```

try...finally

The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data center through the network or working with a file or working with a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource once used, whether it was successful or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee execution.

Here is an example of file operations to illustrate this. **Example 4(Excep 4.py)**

```
try:
   f = open("test.txt",”w”,encoding = 'utf-8')
   # perform file operations
finally:
   f.close()
```

Python Custom Exceptions

Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.

However, sometimes you may need to create custom exceptions that serves your purpose.

In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived from this class.

Example 3(Excep 3.py)

```
# define Python user-defined exceptions
class Error(Exception):
   """Base class for other exceptions"""
   pass

class ValueTooSmallError(Error):
   """Raised when the input value is too small"""
   pass

class ValueTooLargeError(Error):
   """Raised when the input value is too large"""
   pass

# our main program
# user guesses a number until he/she gets it right

# you need to guess this number
number = 10

while True:
   try:
       i_num = int(input("Enter a number: "))
       if i_num < number:
           raise ValueTooSmallError
       elif i_num > number:
           raise ValueTooLargeError
       break
   except ValueTooSmallError:
       print("This value is too small, try again!")
       print()
   except ValueTooLargeError:
       print("This value is too large, try again!")
       print()

print("Congratulations! You guessed it correctly.")
```

Output is:

```
Enter a number: 12
This value is too large, try again!

Enter a number: 0
This value is too small, try again!

Enter a number: 8
This value is too small, try again!

Enter a number: 10
```

Congratulations! You guessed it correctly.

Here, we have defined a base class called Error.

The other two exceptions (ValueTooSmallError and ValueTooLargeError) that are actually raised by our program are derived from this class. This is the standard way to define user-defined exceptions in Python programming,