Python Exception Handling - Try, Except and Finally

When an exception occurs in Python, 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.

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

while True:
   try:
       x = int(input("Enter an integer: "))
       r = 1/x
       break
   except:
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again.")
       print()

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

Here is a sample run of this program.

```
Enter an integer: a
Oops! <class 'ValueError'> occured.
Please try again.

Enter an integer: 1.3
Oops! <class 'ValueError'> occured.
Please try again.

Enter an integer: 0
```

```
Oops! <class 'ZeroDivisionError'> occured.
Please try again.

Enter an integer: 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' and '1.3' 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.

```
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.

```
>>> raise KeyboardInterrupt
Traceback (most recent call last):
...
KeyboardInterrupt

>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument

>>> 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 to illustrate this.

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

This type of construct makes sure the file is closed even if an exception occurs.