3.1 EXCEPTIONS HANDLING

The simplest way to handle exceptions is with a "try-except" block. Exceptions that are caught in try blocks are handled in except blocks. If an error is encountered, a try block code execution is stopped and control transferred down to except block.

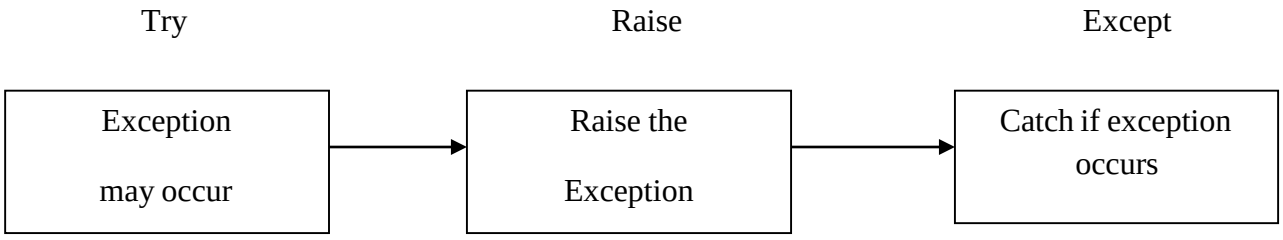

Fig: Exception Handling

Syntax:

try:

statements

Except exception1:

If exception1 occurs execute it.

Except exception2:

If exception2 occurs execute it.

The **try** statement works as follows.

i) First, the **try** block (the statement(s) between the try and except keywords) is executed.

ii) If no exception occurs, the **except** block is skipped and execution of the try statement is finished.

iii) If an exception1 occurs rest of try block is skipped, except block1 gets executed.

iv) If an exception2 occurs rest of try block is skipped, except block2 gets executed.

A simple example to handle divide by zero error is as follows.

```
(x,y) = (5,0)
try:
    z = x/y
except ZeroDivisionError:
print "divide by zero"
```

Output:

```
divide by zero
```

A try statement may have more than one except clause, to specify handlers for different exceptions. If an exception occurs, Python will check each except clause from the top down to see if the exception type matches. If none of the except clauses match, the exception will be considered unhandled, and your program will crash.

Syntax:

```
try:
    # statements
    break
except ErrorName1:
    # handler code
except ErrorName2:
    # handler code
```

A simple example to handle multiple exceptions is as follows.

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("%d / %d = %f" % (dividend, divisor, dividend/divisor))
except ValueError:
    print("The divisor and dividend have to be numbers!")
```

```
except ZeroDivisionError:
        print("The dividend may not be zero!")
```

Output (successful):

```
Please enter the dividend: 12
Please enter the divisor: 2
12 / 2 = 6.000000
```

Output (unsuccessful-divide by zero error):

```
Please enter the dividend: 100
Please enter the divisor: 0
The dividend may not be zero!
```

Output (unsuccessful-value error):

```
Please enter the dividend: 'e'
The divisor and dividend have to be numbers!
```

An except clause may name multiple exceptions as a parenthesized tuple, for example:

```
... except (RuntimeError, TypeError, NameError):
...#handlercode
```

Example:

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("%d / %d = %f" % (dividend, divisor, dividend/divisor))
except(ValueError, ZeroDivisionError):
     print("Oops, something went wrong!")
```

Output:

```
Please enter the dividend: 110
Please enter the divisor: 0
Oops, something went wrong!
```

To catch all types of exceptions using single except clause, simply mention except keyword without specifying error name. It is shown in following example.

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("%d / %d = %f" % (dividend, divisor, dividend/divisor))
except:
    print("Oops, something went wrong!")
```

3.3.1 Raising exceptions

The raise statement initiates a new exception. It allows the programmer to force a specified exception to occur. The raise statement does two things: it creates an exception object, and immediately leaves the expected program execution sequence to search the enclosing *try* statements for a matching except clause. It is commonly used for raising user defined exceptions. Syntax of two forms of the raise statement are:

```
raise ExceptionClass(value)
raise Exception
```

Example:

```
try:
    raise NameError
except NameError:
    print('Error')
```

Output:

Error

raise without any arguments is a special use of python syntax. It means get the exception and re-raise it. The process is called as re-raise**.** If no expressions are present, raise re-raises the

last exception that was active in the current scope.

Example:

```
try:
        raise NameError
except NameError:
        print('Error')
        raise
```

Output:

```
Error
Traceback (most recent call last):
File "main.py", line 2, in <module>
raise NameError('Hi')
NameError: Hi
```

In the example, raise statement inside except clause allows you to re-raise the exception NameError.

3.3.2The else and finally statements

Two clauses that can be added optionally to try-except block are **else** and **finally**. **else** will be executed only if the try clause doesn't raise an exception.

```
try:
        age = int(input("Please enter your age: "))
except ValueError:

        print("Value error occured")

else:
        print("I see that you are %d years old." % age)
```

Output:

```
Please enter your age:10
I see that you are 10 years old.
Please enter your age:'a'
```

Value error occured

3.3.3 User-defined exceptions

Python allows the user to create their custom exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class.

```
# define Python user-defined exceptions
class Error(Exception):
    """Base class for other exceptions"""
    pass # null operation
class PosError(Error):
    """Raised when the input value is positive"""
    pass
class NegError(Error):
    """Raised when the input value is negative"""
    Pass
```

3.4 MULTIPLE EXCEPTIONS

Different Exceptions:

Python helps with several built-in exceptions, and the following is a list of some common exceptions that a standard Python program can throw.

- EOFError

  This error occurs when the end of the file is reached, but still, some operations are left to perform.
- IOError

  This error occurs when the Input Output operation fails.
- ZeroDivisionError

  This error occurs when a number is divided by zero.
- NameError

  This error occurs when the python script is not able to find a name.
- IndentationError

  This error occurs when the Input Output operation fails.

Handling multiple exceptions

Example 1

One can handle different exceptions by using a single block of code, which can be grouped in a tuple, as shown in the example below.

Code:

```
try:
client_obj.get_url(url)
except (URLError, ValueError, SocketTimeout):
    client_obj.remove_url(url)
```

Explanation:

The remove_url() method will have to be called if any of the listed exceptions occur.

Example 2

On the other hand, if any exceptions have to be handled separately, then the following code could help us.

Code:

```
try:
    client_obj.get_url(url)
except (URLError, ValueError):
    client_obj.remove_url(url)
except SocketTimeout:
    client_obj.handle_url_timeout(url)
```

Explanation:

Here we separated the two exceptions, and thus, they can be handled separately now.

Example 3

Now instead of putting exceptions grouped into an inheritance hierarchy, all the exceptions can be caught simply by specifying a base class, like in the code given below.

Code:

```
try:
    f = open(filename)
except (FileNotFoundError, PermissionError):
```

Explanation:

Here in the above code, the except statement follows a hierarchy, and we can omit this by using the given code below.

Code:

```
try:
    f = open(filename)
except OSError:
```

Explanation:

The OSError mentioned above is a base class that is common to both the FileNotFoundError and also the PermissionError exceptions. Hence, it can be used to replace the hierarchy easily.

Example 4

It is unnecessary to handle multiple exceptions, and hence one can get a handle on the thrown exception using them as a keyword as used in the given code.

Code:

```
try:
    f = open(filename)


except OSError as e:
    if e.errno == errno.ENOENT:
        logger.error('File not found')
    elif e.errno == errno.EACCES:
        logger.error('Permission denied')
    else:
        logger.error('Unexpected error: % d', e.errno)
```

Output:

```
File not found
```

Explanation:

The ‘e’ variable used above basically holds an instance of the raised OSError. This can prove useful if the exception has to be invested even further, maybe to process it based on the value of the additional status code.The clauses of the except are checked in the order list, and the first match is executed.

You can try it on online python compiler.