Exception Handling

In any programming language there are 2 types of errors are possible.

1. Syntax Errors
2. Runtime Errors

1. Syntax Errors:

The errors which occurs because of invalid syntax are called syntax errors.

Note:
Programmer is responsible to correct these syntax errors. Once all syntax errors are corrected then only program execution will be started.

2. Runtime Errors:

Also known as exceptions.
While executing the program if something goes wrong because of end user input or programming logic or memory problems etc then we will get Runtime Errors.

Note: Exception Handling concept applicable for Runtime Errors but not for syntax errors

What is Exception:

An unwanted and unexpected event that disturbs normal flow of program is called exception.

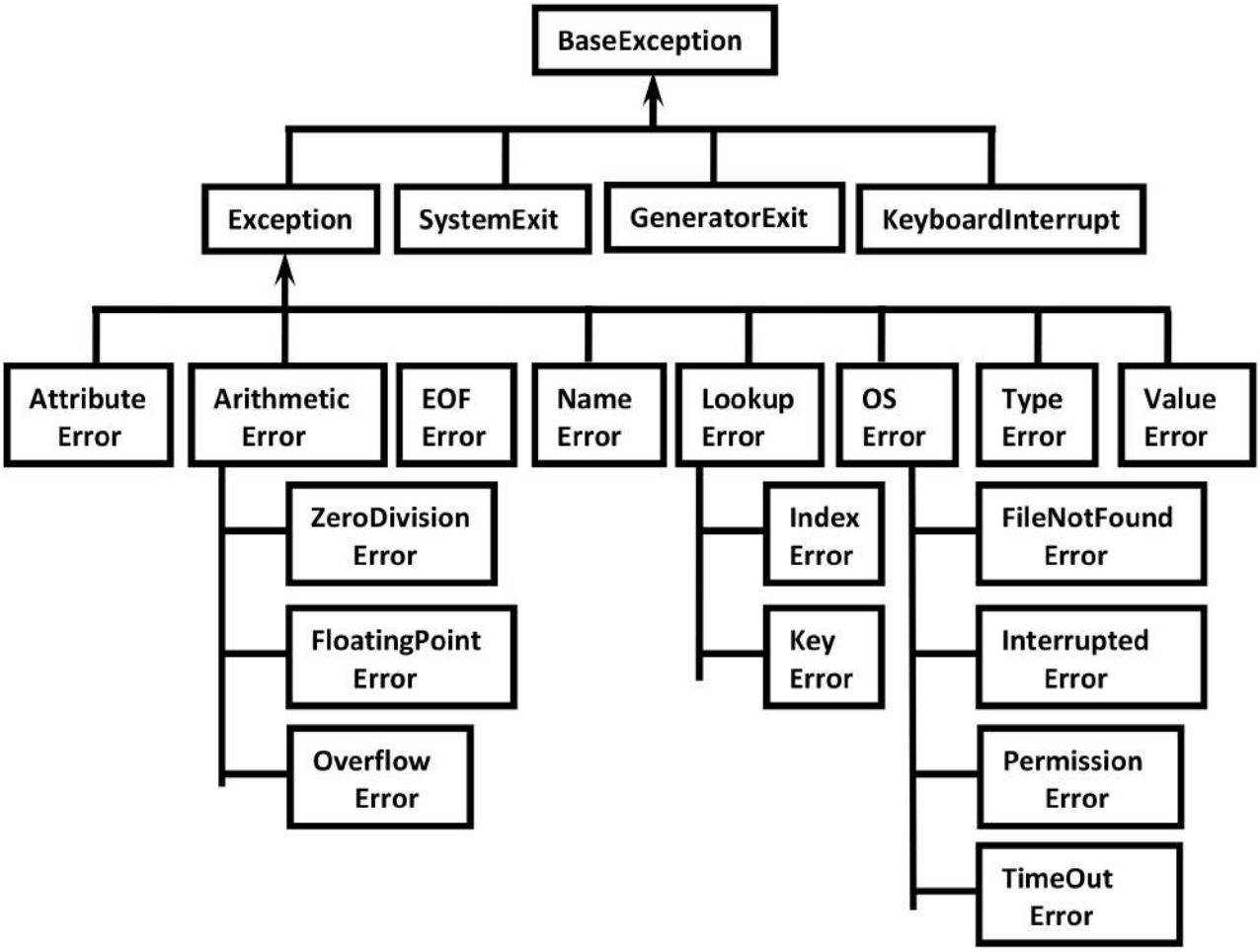
Eg:

ZeroDivisionError
TypeError
ValueError
FileNotFoundError
EOFError
SleepingError
TyrePuncturedError

It is highly recommended to handle exceptions. The main objective of exception handling is Graceful Termination of the program(i.e we should not block our resources and we should not miss anything)

Exception handling does not mean repairing exception. We have to define alternative way to continue rest of the program normally.

Python's Exception Hierarchy



Every Exception in Python is a class.
All exception classes are child classes of BaseException.i.e every exception class extends BaseException either directly or indirectly. Hence BaseException acts as root for Python Exception Hierarchy.

Most of the times being a programmer we have to concentrate Exception and its child classes.

Customized Exception Handling by using try-except:

It is highly recommended to handle exceptions.
The code which may raise exception is called risky code and we have to take risky code inside try block. The corresponding handling code we have to take inside except block.

try:
Risky Code
except XXX:
Handling code/Alternative Code

Control Flow in try-except:

```
try:
        stmt-1
        stmt-2
        stmt-3
except XXX:
      stmt-4
stmt-5
```

case-1: If there is no exception
1,2,3,5 and Normal Termination

case-2: If an exception raised at stmt-2 and corresponding except block matched
1,4,5 Normal Termination

case-3: If an exception raised at stmt-2 and corresponding except block not matched
1, Abnormal Termination

case-4: If an exception raised at stmt-4 or at stmt-5 then it is always abnormal termination.

Conclusions:

1. within the try block if anywhere exception raised then rest of the try block wont be executed eventhough we handled that exception. Hence we have to take only risky code inside try block and length of the try block should be as less as possible.

2. In addition to try block,there may be a chance of raising exceptions inside except and finally blocks also.

3. If any statement which is not part of try block raises an exception then it is always abnormal termination.

try with multiple except blocks:

The way of handling exception is varied from exception to exception.Hence for every exception type a seperate except block we have to provide. i.e try with multiple except blocks is possible and recommended to use.

Single except block that can handle multiple exceptions:

We can write a single except block that can handle multiple different types of exceptions.

except (Exception1,Exception2,exception3,..): or
except (Exception1,Exception2,exception3,..) as msg :

Parenthesis are mandatory and this group of exceptions internally considered as tuple.

Default except block:

We can use default except block to handle any type of exceptions.
In default except block generally we can print normal error messages.

Syntax:

```
except:
    statements
```

***Note: If try with multiple except blocks available then default except block should be last,otherwise we will get SyntaxError.

Note:

The following are various possible combinations of except blocks

1. except ZeroDivisionError:
1. except ZeroDivisionError as msg:
3. except (ZeroDivisionError,ValueError) :
4. except (ZeroDivisionError,ValueError) as msg:
5. except :

finally block:

1. It is not recommended to maintain clean up code(Resource Deallocating Code or Resource Releasing code) inside try block because there is no guarentee for the execution of every statement inside try block always.

2. It is not recommended to maintain clean up code inside except block, because if there is no exception then except block won't be executed.

Hence we required some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether exception handled or not handled. Such type of best place is nothing but finally block.

Hence the main purpose of finally block is to maintain clean up code.

```
try:
    Risky Code
except:
    Handling Code
finally:
    Cleanup code
```

The speciality of finally block is it will be executed always whether exception raised or not raised and whether exception handled or not handled.

*** **Note:** There is only one situation where finally block won't be executed ie whenever we are using os._exit(0) function.

Whenever we are using os._exit(0) function then Python Virtual Machine itself will be shutdown.In this particular case finally won't be executed.

Note:

os._exit(0)

where 0 represents status code and it indicates normal termination

There are multiple status codes are possible.

else block with try-except-finally:

We can use else block with try-except-finally blocks.
else block will be executed if and only if there are no exceptions inside try block.

```
try:
    Risky Code
except:
    will be executed if exception inside try
else:
    will be executed if there is no exception inside try
finally:
    will be executed whether exception raised or not raised and handled or not handled
```

Types of Exceptions:

In Python there are 2 types of exceptions are possible.

1. Predefined Exceptions
2. User Definded Exceptions

1. Predefined Exceptions:

Also known as in-built exceptions

The exceptions which are raised automatically by Python virtual machine whenver a particular event occurs, are called pre defined exceptions.

2. User Defined Exceptions:

Also known as Customized Exceptions or Programatic Exceptions

Some time we have to define and raise exceptions explicitly to indicate that something goes wrong ,such type of exceptions are called User Defined Exceptions or Customized Exceptions

Programmer is responsible to define these exceptions and Python not having any idea about these. Hence we have to raise explicitly based on our requirement by using "raise" keyword.

How to Define and Raise Customized Exceptions:

Every exception in Python is a class that extends Exception class either directly or indirectly.

Syntax:

```
class classname(predefined exception class name):
    def __init__(self,arg):
        self.msg=arg
```

Note:

raise keyword is best suitable for customized exceptions but not for pre defined exceptions

PYTHON DEBUGGING BY USING ASSERTIONS

Debugging Python Program by using assert keyword:

Types of assert statements:

There are 2 types of assert statements

1. Simple Version:

assert conditional_expression

2. Augmented Version:

assert conditional_expression,message