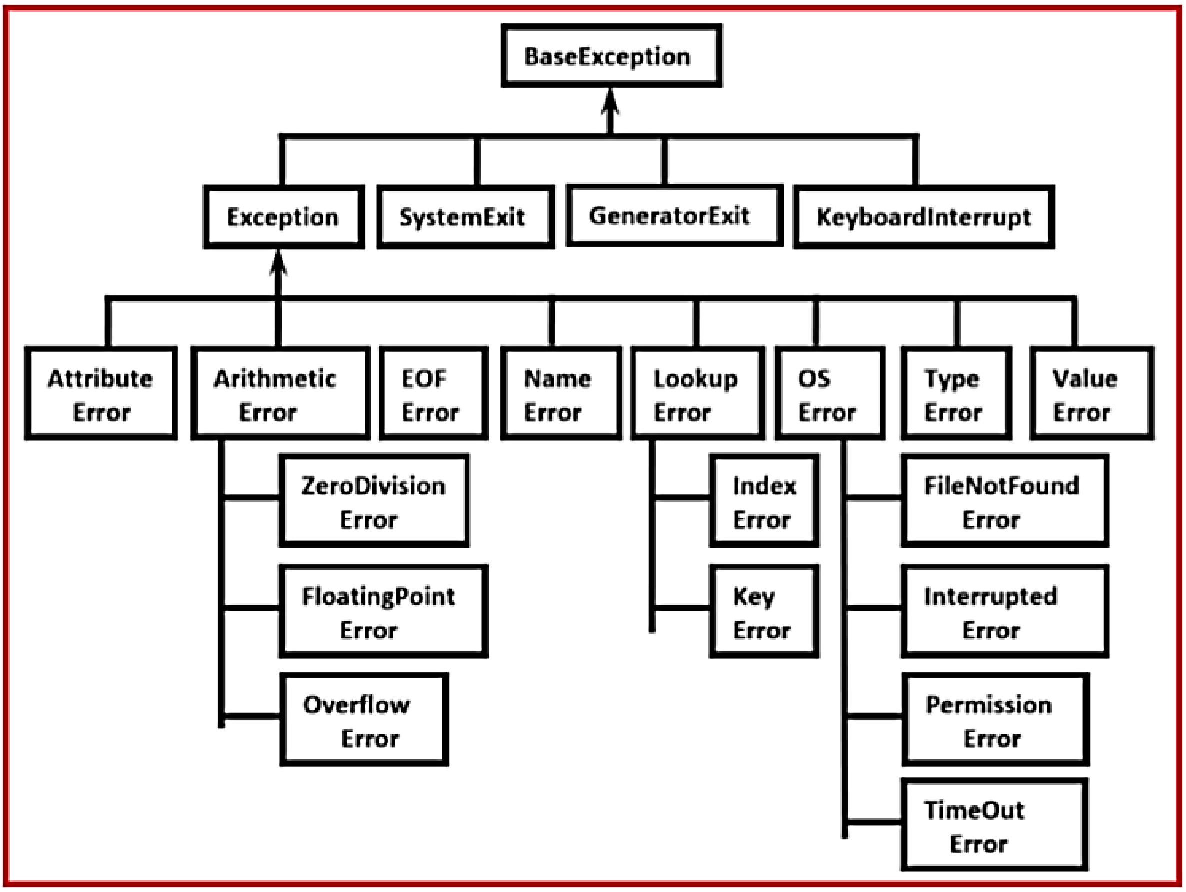
Exception Handling

In our application there are chances of unwanted situation by which our normal flow of program gets disturbed. Sometimes due to it our application gets terminate abnormally. For avoiding abnormal termination or normal flow disturbance we need to use Exception handling.

An exception is a Python object that represents an error. Exception handling results in normal termination of our application by displaying proper messages for particular error. Exception always occurs at run time. There are several reasons because of which exception will occur like, wrong user input, logical code mistake by developer, memory shortage etc.

Python has many **built-in exceptions** that enable our program to run without interruption and give the output. These exceptions are given below in diagram,



Exception Class Hierarchy

Without Exception handling:

```
print('Program starts')
print(10/0)
print('Program ends')
```

OUTPUT:

```
Program starts
ZeroDivisionError: division by zero
```

The try-except block

If the Python program contains suspicious code that may throw the exception, we must place that code in the **try** block. The **try** block must be followed with the **except** statement, which contains a block of code that will be executed if there is some exception in the try block.

Syntax:

```
try:
        ---
        doubtable code in which exception can occur
        ---
except:
        ---
        code to be executed when exception occurs/Error message
```

With exception handling:

```
try:
        print('Program starts')
        print(10/0)
        print('Program ends')
except:
        print('except block')
```

OUTPUT:

```
Program starts
except block
Program ends
```

except block should be followed by problem related exception class or their parent exception class only. If we write other exception class then program will terminate abnormally. If there is no exception in try block then execution of except block will be skipped.

✓

```
try:
        print('Program starts')
        print(10/0)
        print('Program ends')
except ZeroDivisionError:
        print('except block')
```

OUTPUT:

```
Program starts
except block
Program ends
```

✗

```
try:
        print('Program starts')
        print(10/0)
        print('Program ends')
except ValueError:
        print('except block')
```

OUTPUT:

```
Program starts
ZeroDivisionError: division by zero
```

Multiple except block:

When there is a chance that more than one type of error can come in program then we should display proper message for each error so in this case we use multiple except block. For e.g.,

```
try:
    print('try starts')
    num = int(input('Enter a number : '))
    print(10/num)
    print('try ends')

except ZeroDivisionError:
    print('Please enter a non-zero number')

except ValueError:
    print('Please enter integer only')
```

Scenario 1:

```
try starts
Enter a number : a
Please enter integer only
```

Scenario 2:

```
try starts
Enter a number : 0
Please enter a non-zero number
```

else block:

else block is executed when there is no exception in try block. When exception is there in try block then else block execution is skipped.

Syntax:

```
try:
        ---
        doubtable code in which exception can occur
except:
        ---
        code to be executed when exception occurs/Error message
else:
        ---
        code to be executed when exception does not occur
```

```
try:
        print('Program starts')
        print(10/0)
        print('Program ends')
except ZeroDivisionError:
        print('except block')
else:
        print('else block')
```

OUTPUT:

```
Program starts
except block
```

```
try:
        print('Program starts')
        print(10/2)
        print('Program ends')
except ZeroDivisionError:
        print('except block')
else:
        print(‘else block')
```

OUTPUT:

```
Program starts
5.0
Program ends
else block
```

Finally block:

- It is always executed either there is a problem inside try block or not.
- Code which is needed to be executed compulsory should be written inside it.
- Resources which are open inside try block should be closed inside finally.

For e.g., File closing and database connection closing part should be written inside finally block.

Syntax:

```
try:
        ----block of code which may throw exception-----
finally:
          ----block of code which will always be executed-----
```

#With exception

```
try:
   print('try starts')
   print(10/0)
   print('try ends')
finally:
   print('finally block')
```

OUTPUT:

```
try starts
finally block
ZeroDivisionError: division by zero
```

#Without exception

```
try:
   print('try starts')
   print(10/2)
   print('try ends')
finally:
   print('finally block')
```

OUTPUT:

```
try starts
5.0
try ends
finally block
```

Scenarios of finally with return statement

Scenario 1: If try with finally block and try have a return statement then before control return back to caller, first of all finally block code will be executed.

```
def m1():
    x = 10
    try:
        print('try block')
        return x
    finally:
        print('finally block')

x = m1()
print(x)
```

OUTPUT:

```
try block
finally block
10
```

Scenario 2: If try with finally block and try have a return statement then finally block try to change return statement value then it will be changed only for finally block. It will be not be changed for caller.

```
def m1():
    x = 10
    try:
        print('try block')
        return x
    finally:
        x = 20
        print('finally block',x)

x = m1()
print(x)
```

OUTPUT:

```
try block
finally block 20
10
```

Scenario 3: If try and finally both have return statement then finally block return statement will be executed.

```
def m1():
    x = 10
    try:
        print('try block')
        return x
    finally:
        x = 20
        print('finally block',x)
        return x

x = m1()
print(x)
```

OUTPUT:

```
try block
finally block 20
20
```

Exception propagation:

It means give a chance to the caller function to handle the exception.

For e.g.,

```
def m1():
    print('m1 starts')
    print(10/0)
    print('m1 ends')

def m2(): →caller function as inside it m1 function is called
    print('m2 starts')
    try:
        m1()
    except ZeroDivisionError:
        print('except block')
    print('m2 ends')
m2()
```

OUTPUT:

```
m2 starts
m1 starts
except block
m2 ends
```

raise keyword:

It is used to occur our own exception. Using it we can also occur custom exception. By raise keyword, we can raise built-in as well as custom exception class. raise keyword is always followed by exception class object.
Custom exception class should be inherited from Exception class. For e.g.,

```
class AgeInvalidError(Exception):  →Custom exception class
    def __init__(self,msg):
        super().__init__(msg)

def m1(age):
    if age < 0:
        raise AgeInvalidError('Age cannot be negative!!')

try:
    m1(int(input('Enter age : ')))
except AgeInvalidError as message:
    print('Error message : ',message)
```

OUTPUT:

```
Enter age : -2
Error message :  Age cannot be negative!!
```