03-Python intermediate

Exception Handling

Posted on September 19, 2021
Last updated on September 19, 2021

3. Exception Handling

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

1. Syntax Errors
2. Runtime Errors

3.1 Syntax Errors

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

```
x=10
if x==10
print("Hello")

# SyntaxError: invalid syntax
```

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

3.2 Runtime Errors

`AKA 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.

```
print(10/0)
# ZeroDivisionError: division by zero

print(10/"ten")
# TypeError: unsupported operand type(s) for /: 'int' and 'str
```

3.3 Exception

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

It is `highly recommended` to handle exceptions. The main objective of exception handling is Graceful Termination of the program

To handle exception and proper execution of program we put Risky code inside `try-except` block

try-except

```
try:
    Risky Code
except XYZ:
    Handling code/Alternative Code
```

- Default except block - handles all exception

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

- try with multiple except blocks

```
try:
    Risky Code
except ZeroDivisionError:
    Handling code/Alternative Code
except FileNotFoundError:
    Handling code/Alternative Code
```

- Single except block that can handle multiple exceptions

```
try:
    Risky Code
except (Exception1,Exception2,exception3,..):
    Handling code/Alternative Code

# except (Exception1,Exception2,exception3,..) as msg
```

try-except-finally

finally block is `executed always` irrespective of whether exception raised or not raised and whether exception handled or not handled.

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

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

3.4 Custom Exception

We can define custom exception by using `"raise" keyword`

```
age = -1
if age < 1:
    raise RuntimeError("Wrong age")
else:
    print("Done")
```

```
class NegativeAgeException(Exception):
    def __init__(self,arg):
        self.msg=arg

age = -1
if age < 1:
    raise NegativeAgeException("Age can't be Negative")
else:
    print("Valid Age")
```

3.5 List Of General Use Exceptions

```
try :
    a = 10
    10/0
except ZeroDivisionError as e :
    print(e)
# division by zero
```

```
try :
    int("sudh")
except (ValueError , TypeError) as e :
    print(e)
# invalid literal for int() with base 10: 'sudh'
```

```
# Not good practice
# Shoulld use always a specific exception

try :
    int("sudh")
except  :
    print("this will catch an error")
# this will catch an error


try :
    with open("test.txt" , 'r') as f :
        f.read()
except Exception as e :
    print("test " , e)
except FileNotFoundError as e :
    print("this is my file not found type error " , e)

# test  [Errno 2] No such file or directory: 'test.txt'
# last `except block` will never get executed
```

```
try :
    import sudh
except ImportError as e :
    print(e)
# No module named 'sudh'
```

```
try :
    d = {1: [3,4,5,6] , "key" :"sudh"}
    d["key10"]
except KeyError as e :
    print(e)
# 'key10'
```

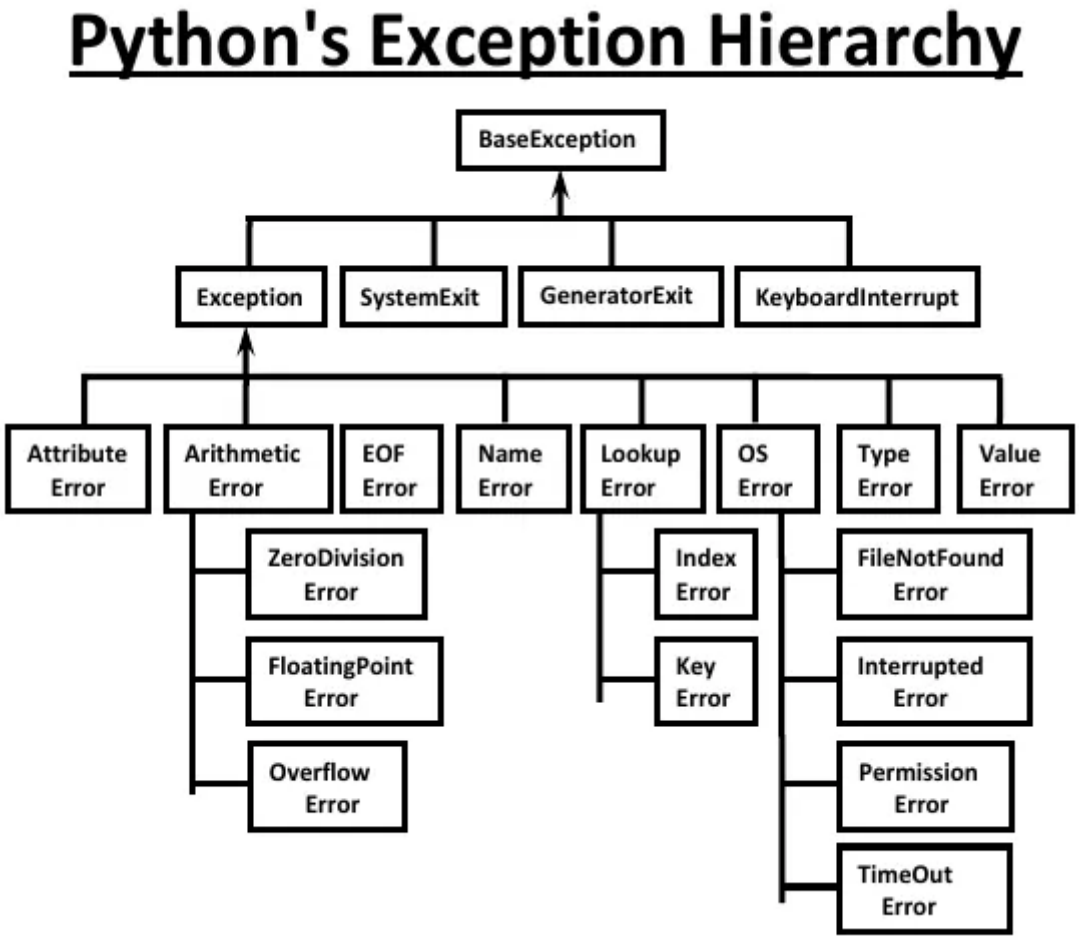
```
try :
    "sudh".test()
except AttributeError as e :
    print(e)
# 'str' object has no attribute 'test'
```

```
try :
    l = [1,2,3,3]
    l[10]
except IndexError as e :
    print(e)
# list index out of range
```

```
try :
    123 + "sudh"
except TypeError as e :
    print(e)
# unsupported operand type(s) for +: 'int' and 'str'
```

```
try :
    with open("test.txt" , 'r') as f :
        f.read()
except FileNotFoundError as e :
    print(e)
# [Errno 2] No such file or directory: 'test.txt'
```

3.6 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.

```
# Print Python Exception Hierarchy
def treeClass(cls, ind=0):
    # print name of the class
    print('-' * ind, cls.__name__)

    # iterating through subclasses
    for i in cls.__subclasses__():
        treeClass(i, ind + 3)

print("Hierarchy for Built-in exceptions is : ")
treeClass(BaseException)
```

.

.

.

.

.

.

.

.

.

.

.

.

.

.