Inner classes or nested classes

A class defined inside class is called inner class or nested class.

What is use of inner classes?

1. Hiding class
2. Reusability

Inner classes are two types

1. Member class
2. Local class

Member class

If a class defined inside class and outside method is called member class. This member class can be private, public or protected.

Syntax:

```
class <outer-class-name>:
        class <member-class-name>:
                variables
                methods
        methods
```

Example:

```
class A:  # Outer class
    class B: # Member class/public
        def m1(self):
            print("m1 of B")
    class __C: # Member class/private
        def m2(self):
            print("m2 of C")


objb=A.B()
objb.m1()
```

Output

m1 of B

Example:

```
class Person:
    class __Address:
        def __init__(self):
            self.__street=None
            self.__city=None
        def read_address(self):
            self.__street=input("Enter Street :")
            self.__city=input("Enter City :")
        def print_address(self):
            print(f'Street {self.__street}')
            print(f'City {self.__city}')
    def __init__(self):
        self.__name=None
        self.__add1=Person.__Address()
        self.__add2=Person.__Address()
    def read_person(self):
        self.__name=input("Enter Name ")
        self.__add1.read_address()
        self.__add2.read_address()
    def print_person(self):
        print(f'Name {self.__name}')
        self.__add1.print_address()
        self.__add2.print_address()


p1=Person()
p1.read_person()
p1.print_person()
```

Output

```
Enter Name naresh
Enter Street :ameerpet
Enter City :hyd
Enter Street :s.r.nager
Enter City :hyd
Name naresh
```

Street ameerpet
City hyd
Street s.r.nager
City hyd

Local class

A class defined inside method is called local class.
This class is used within method but cannot used outside the method.

Syntax:

```
class <outer class-name>
        def <method-name>(self):
                class <local-class>:
                        variables
                        methods
```

Example:

```
class Outer:
    def m1(self):
        class Local:
            def m2(self):
                print("m2 of Local class")
        print("m1 of outer class")
        obj=Local()
        obj.m2()

obj1=Outer()
obj1.m1()
```

Output

m1 of outer class
m2 of Local class

OOP

1. Encapsulation
2. Polymorphism
3. Inheritance

4. Class
5. Object
6. Abstraction

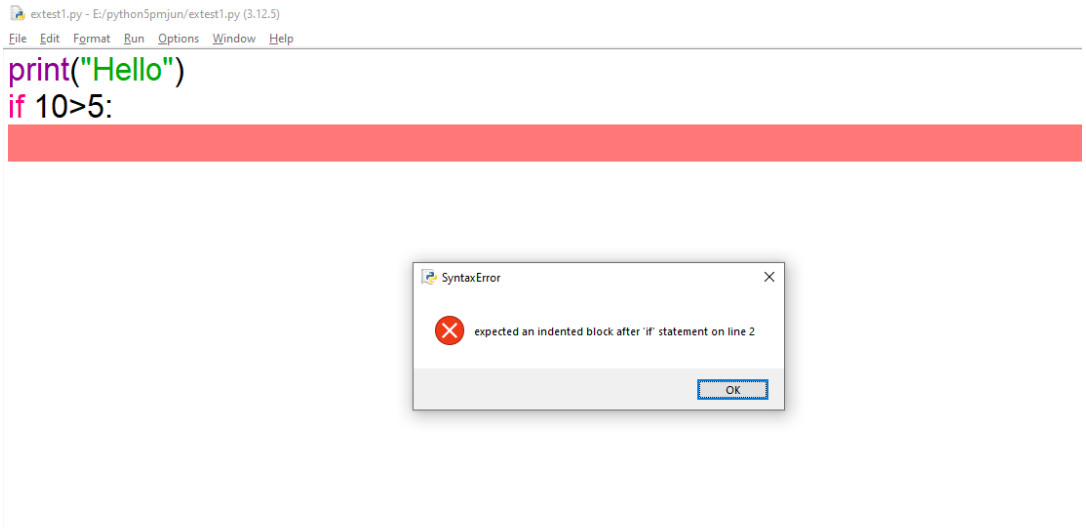
Exception Handling or Error Handling

Types of Errors

1. Compile time Errors
2. Logical Errors
3. Runtime Errors

Compile time errors

An error which occurs during compiling of program is called compile time error, all syntax errors are called compile time errors.
If there is a compile time error within program, program execution is stopped.
The syntax errors must be rectified by programmer.



Logical Errors

Logic is nothing but a group of statements used to solve given problem. If group of statements are not written properly program display logical error. When there is logical error within program, program gives wrong result.

Example:

```
num1=int(input("Enter First Number "))
num2=int(input("Enter Second Number "))
if num1>num2:
    print(num2,"is max")
else:
    print(num1,"is max")
```

Output

```
Enter First Number 5
Enter Second Number 2
2 is max
```

These logical errors must rectified by the programmer in order to get accurate result.

Runtime Errors

The errors which occur during execution of program are called runtime errors. Runtime errors occur because of wrong input given by end user.

When there is runtime error within program, program execution is terminated or stopped.

To avoid abnormal termination of program, we use error handlers or exception handlers.

What is exception?

Exception is a runtime error.
An error which occurs during execution of program is called exception or runtime error.
Exception occurs because of wrong input given by end user.

Advantage of exception handling or error handling

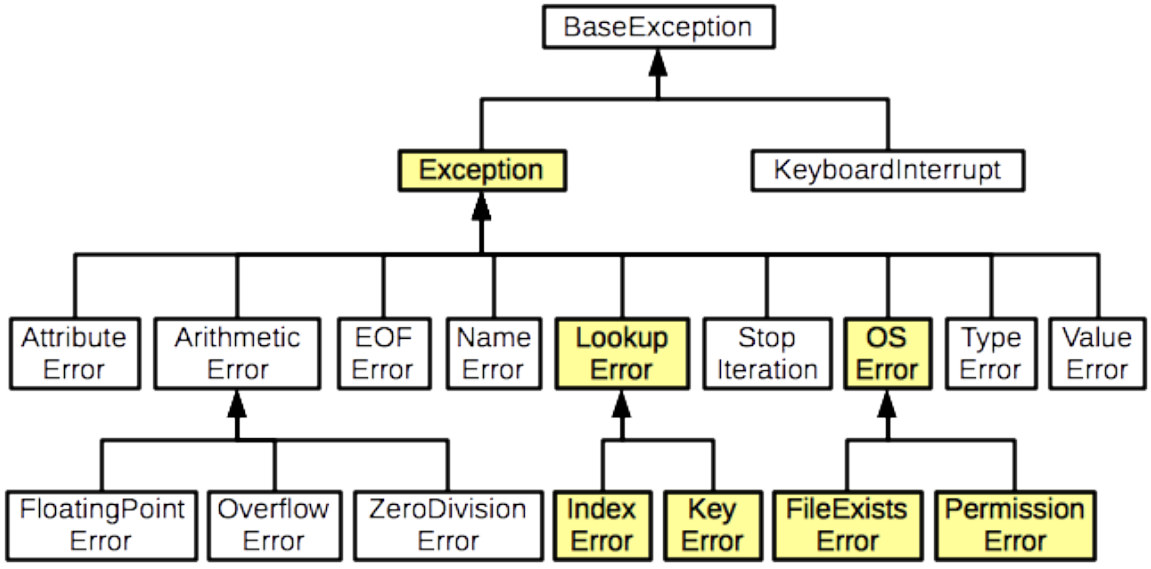
1. Avoiding abnormal termination of program
2. Converting predefined error messages into user defined error messages
3. Separate business logic and error logic

Keywords used in error handling or exception handling

1. try
2. except
3. finally
4. raise
5. assert

Every error is one data type or class
Creating object of error class and giving to PVM (Python Virtual Machine) is called raising an error. A function or method generates runtime when given wrong input.

Hierarchy of Error classes



All predefined error types are used by python and python libraries.
try keyword or block

try block contains, the statements which generate error during runtime (OR) the statements which has to monitored for error handling or exception handling are included within try block.

Syntax:

```
try:
  statement-1
  statement-2
```

except block or keyword
if there is an error or exception within try block, it is handled by except block. A try block followed by one or more than one except block.

```
try:
   statement-1
   statement-2
except <error-type>:
   print user defined error message
except <error-type>:
   print user defined error message
```

Example:

```
# write a program to divide two integers

while True:
    n1=int(input("Enter first integer value "))
    n2=int(input("Enter second integer value "))
    try:
        n3=n1/n2
        print(f'Division of {n1}/{n2} is {n3}')
        break
    except ZeroDivisionError:
        print("cannot divide number with zero try again...")
```

Output

```
Enter first integer value 3
Enter second integer value 0
cannot divide number with zero try again...
Enter first integer value 5
Enter second integer value 2
Division of 5/2 is 2.5
```