

## **Inner classes or Nested classes**

Defining class inside class is called nested class or inner class.

### **Advantage of nested classes**

- 1. Providing security**
- 2. Reusability and Modularity**

### **Types of inner classes**

1. Member class
2. Local class

If a class is defined inside class, it is called member class. This member class is used within outer class but not outside outer class.

Member class can be defined as private, protected or public. Private member class is used within outer class.

Public member class is used within outer class and outside outer class.

Protected member class is used within outer class and within inherited class.

### **Example:**

```
class A: # outer class
    class B: # inner class/member class
        def m1(self):
            print("m1 of B class")
    def m2(self):
        print("m2 of A class")
```

```
def main():
    objb=A.B()
    objb.m1()
main()
```

### **Output:**

m1 of B class

### **Example:**

```
class A: # outer class
    class __B: # inner class/member class
```

```

    def m1(self):
        print("m1 of B class")
def __init__(self):
    self.objb=A.__B()
def m2(self):
    print("m2 of A class")

```

```

def main():
    obja=A()
    obja.m2()
    obja.objb.m1()

```

main()

### Output:

```

===== RESTART: F:/python6pmaug/oopstest47.py =====
m2 of A class
m1 of B class

```

### Example:

```

class Person:
    class Address:
        def __init__(self):
            self.street=None
            self.city=None
        def readAddress(self):
            self.street=input("Street")
            self.city=input("City")
        def printAddress(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 readPerson(self):
        self.__name=input("Name ")
        self.__add1.readAddress()

```

```

        self.__add2.readAddress()
    def printPerson(self):
        print(f'Name {self.__name}')
        self.__add1.printAddress()
        self.__add2.printAddress()
def main():
    p1=Person()
    p1.readPerson()
    p1.printPerson()
main()

```

### **Output:**

```

Name naresh
Streetameerpet
Cityhyd
Streets.r.nager
Cityhyd
Name naresh
Street ameerpel
City hyd
Street s.r.nager
City hyd

```

### **Local class**

If a class is defined inside block or method is called local class. This is local to block, it cannot accessed outside the block.

### **Example:**

```

class A: # Outer class
    def m1(self):
        print("inside m1 of A")
        class B: # Local class
            def m2(self):
                print("inside m2 of B")

        objb=B()
        objb.m2()

def main():
    obja=A()

```

```
    obja.m1()  
main()
```

### Output:

```
===== RESTART: F:/python6pmaug/ooptest49.py =====  
inside m1 of A  
inside m2 of B
```

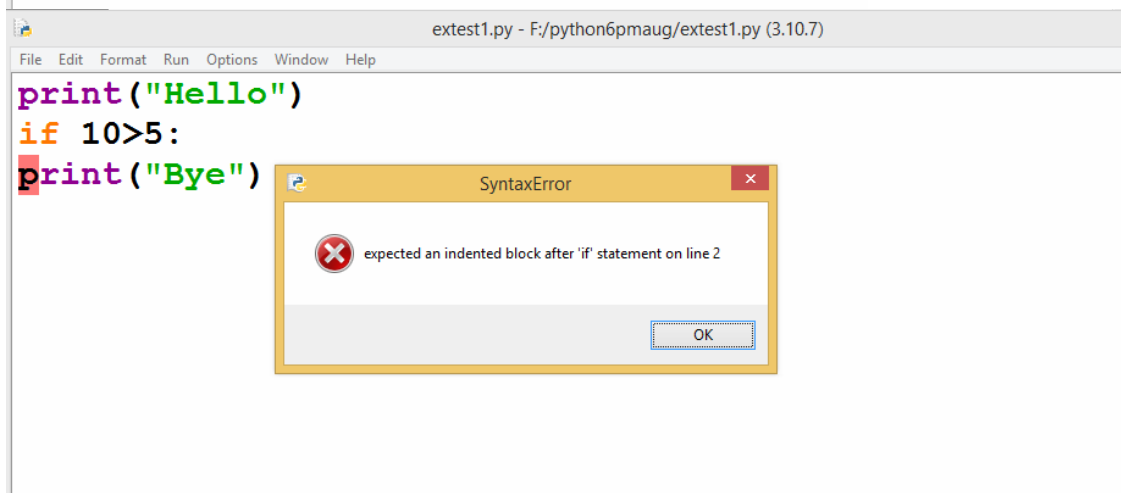
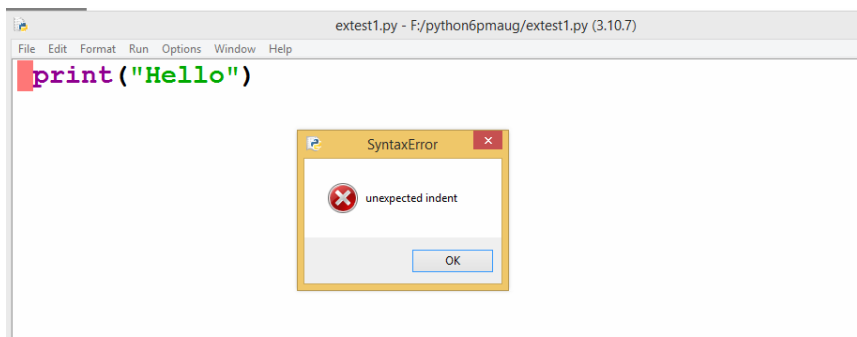
## Exception Handling OR Error Handling

### Types of Errors

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

### Compile Time Errors

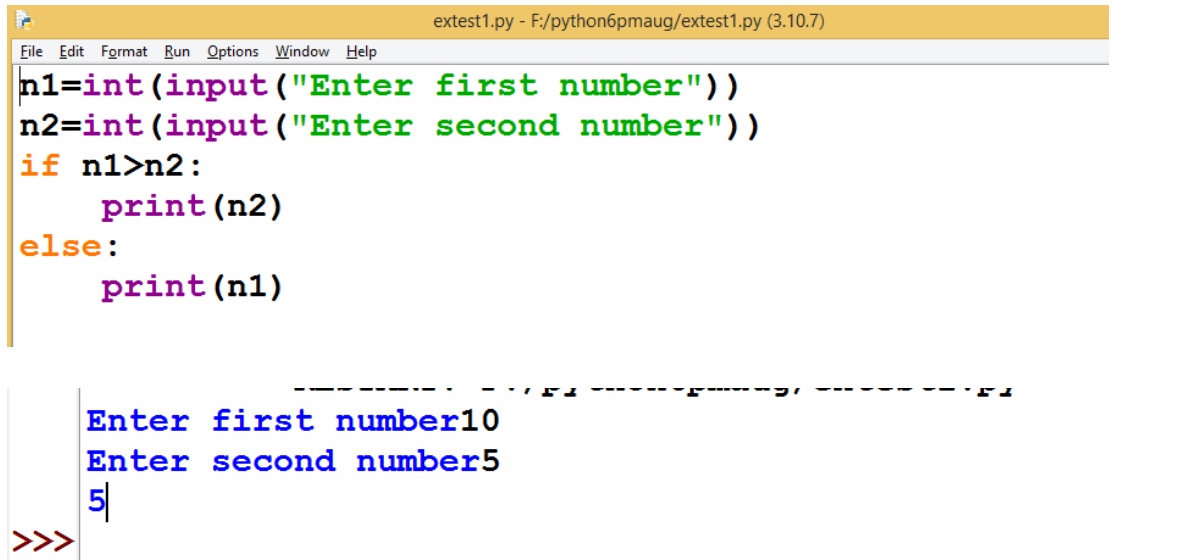
The error which occurs during compiling of program is called compile time errors. All syntax errors are called compile time errors.



If there is a syntax error within program, program execution is not done. These errors have to be rectified in order to execute program.

## Logical Error

If there is an error within logic, it leads to logical error. if there is a logical error within program, program display wrong result/output.



```
extest1.py - F:/python6pmaug/extest1.py (3.10.7)
File Edit Format Run Options Window Help
n1=int(input("Enter first number"))
n2=int(input("Enter second number"))
if n1>n2:
    print(n2)
else:
    print(n1)

Enter first number10
Enter second number5
5
>>>
```

Logical errors must be rectified by programmer, in order to display accurate result.

## Runtime Error

The error which occurs during the execution of program is called runtime error. Runtime error occurs because of wrong input given by end user. When there is a runtime during execution of program, program execution is terminated. All runtime errors are called exceptions. To avoid abnormal termination of program we required exception handlers or error handlers.

### What is need of error handlers or exception handlers?

1. To avoid abnormal termination of program
2. To convert predefined error messages to user defined error messages

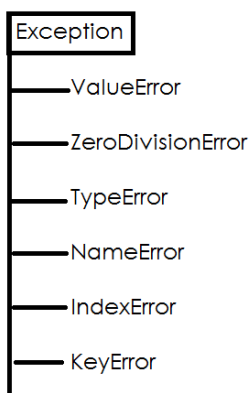
**In order to handle exceptions/errors, python provides the following keywords.**

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

Every error is one type/class, raising an error is nothing creating of error object and giving to python virtual machine. An error is raised by functions or python virtual machine.

Every error object is having message/description.

All error types are inherited from Exception class. Exception is a root class for all error types.



### **try block**

this block contain the statements which has to monitored for exception handler (OR) statements which raises an error during runtime those statements are included within this block.

### **Syntax:**

**try:**

**statement-1**

**statement-2**

### **except block**

except block is error handler block. If there is an error inside try block, that error is handled by except block.

Try block followed by one or more than one except block.

**Syntax:**

```
try:
    statement-1
    statement-2
except <error-type> as <variable-name>:
    statement-3
except <error-type> as <variable-name>:
    statement-4
```

**ZeroDivisionError**

Raised when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.

**Example:**

```
# write a program to divide two numbers
n1=int(input("Enter first number"))
n2=int(input("Enter second number"))
try:
    n3=n1/n2
    print(f'Result is {n3:.2f}')
except ZeroDivisionError:
    print("Cannot divide number with zero")
```

**Output:**

```
===== RESTART: F:/python6pmaug/extest1.py =====
Enter first number4
Enter second number0
Cannot divide number with zero
```

```
===== RESTART: F:/python6pmaug/extest1.py =====
Enter first number5
Enter second number2
Result is 2.50
```

If try block raises more than one type of error, those errors are handled using multiple except blocks.

**Example:**

# write a program to divide two numbers

try:

```
n1=int(input("Enter first number"))
n2=int(input("Enter second number"))
n3=n1/n2
print(f'Result is {n3:.2f}')
```

except ZeroDivisionError:

```
print("Cannot divide number with zero")
```

except ValueError:

```
print("input value must be integer")
```

**Output:**

```
===== RESTART: F:/python6pmaug/extest1.py =====
```

```
Enter first number10
```

```
Enter second number20
```

```
Result is 0.50
```

```
===== RESTART: F:/python6pmaug/extest1.py =====
```

```
Enter first number5
```

```
Enter second number0
```

```
Cannot divide number with zero
```

```
===== RESTART: F:/python6pmaug/extest1.py =====
```

```
Enter first number5
```

```
Enter second numberabc
```

```
input value must be integer
```

except block can be defined without exception type or error type. This except block is able to handle any type error. This except block is called generic except block.

**Example:**

# write a program to divide two numbers

import sys

try:

```
n1=int(input("Enter first number"))
n2=int(input("Enter second number"))
n3=n1/n2
```



```

    print(f'Result is {n3:.2f}')
except:
    t=sys.exc_info()
    print(t[1])

```

### Output:

```

===== RESTART: F:/python6pmaug/extest1.py =====
Enter first number5
Enter second number0
division by zero

```

```

===== RESTART: F:/python6pmaug/extest1.py =====
Enter first numberab
invalid literal for int() with base 10: 'ab'

```

### Example:

# write a program to display the content of file

```

fname=input("Enter File Name")
try:
    f=open(fname,"r")
    s=f.read()
    print(s)
except FileNotFoundError:
    print("Invalid FileName")

```

### Output:

```

===== RESTART: F:/python6pmaug/extest2.py =====
Enter File Namefile1.txt
pythonjavaoracle python is general purpose
programming langauge101

```

```

===== RESTART: F:/python6pmaug/extest2.py =====
Enter File Namefile10.txt
Invalid FileName

```

### finally block

finally is not exception handler, it is block which is executed after execution of try block and except block.

Finally block is used to de-allocate resources allocated within try block.

<pre>try:     statement-1 except &lt;exception&gt;:     statement-2 finally:     statement-3</pre>	<pre>try:     statement-1 finally:     statement-2</pre>	<pre>try:     statement-1 except &lt;exception-1&gt;:     statement-2 except &lt;exception-2&gt;:     statement-3 finally:     statement-4</pre>
--	--	--