finally keyword or finally block

finally block is not an exception handler.
Finally block is used to de-allocate resources allocated within try block
Finally block contains statements which are executed after execution of try block and except block.

Syntax-1:	**Syntax-2:**	**Syntax-3 (Wrong)**

try:
  statement-1
  statement-2
except <error-type>:
  statement-1
finally:
  statement-1
	try:
	  statement-1
	  statement-2
	finally:
	  statement-1
		try:
		  statement-1
		  statement-2
		except:
		  statement-1
		finally:
		  statement-1
		finally:
		  statement-1

Example:

```
try:
    print("Inside try block")
    n1=int(input("Enter First Number :"))
    n2=int(input("Enter Second Number :"))
    n3=n1/n2
    print(f'Result of {n1}/{n2}={n3:.2f}')
except ZeroDivisionError:
    print("inside except block")
finally:
    print("inside finally block")
```

Output

```
Inside try block
Enter First Number :5
Enter Second Number :2
Result of 5/2=2.50
inside finally block

============= RESTART: D:/fspmar5pm/extest4.py ============
Inside try block
```

```
Enter First Number :4
Enter Second Number :0
inside except block
inside finally block
>>>
============= RESTART: D:/fspmar5pm/extest4.py ============
Inside try block
Enter First Number :5
Enter Second Number :abc
inside finally block
Traceback (most recent call last):
  File "D:/fspmar5pm/extest4.py", line 4, in <module>
    n2=int(input("Enter Second Number :"))
ValueError: invalid literal for int() with base 10: 'abc'
```

Finally block is executed,

1. Normal execution of try block (No Error)
2. Handled Error (Error is raised by try block and handled by except block)
3. Unhandled Error (Error is raised by try block but not handled by except block), terminates execution program
4. Forced exit occurs return and break statement

Execution of finally block is must. Finally contains critical statements.

```
try:
   establish connection to database(server)     ==> Resource Allocation
   send SQL statements
   print result
except SQLError:
   print error message
finally:
   close connection    ==> Resource De-Allocation
```

raise keyword

raise keyword is used for generating exception or error.
Generating error is nothing but creating object of error class/exception class and giving to PVM.

A function or method generates error using raise keyword.
A function or method generates an error because wrong input.
This allows separation of business logic and error logic.

It is used when developer develops custom functions or user defined functions.

Syntax:
raise error-class/exception-class()

Example:
```
def multiply(n1,n2):
    if n1==0 or n2==0:
        raise ValueError()
    else:
        return n1*n2


while True:
    a=int(input("Enter first number "))
    b=int(input("Enter second number "))
    try:
        c=multiply(a,b)
        print(f'Product of {a}x{b}={c}')
        break
    except ValueError:
        print("cannot multiply number with zero")
```

Output
```
Enter first number 5
Enter second number 2
Product of 5x2=10

============ RESTART: D:/fspmar5pm/extest5.py ============
Enter first number 6
Enter second number 0
cannot multiply number with zero
Enter first number 8
Enter second number 9
Product of 8x9=72
```

Custom Error Type or User defined Error Types

Creating error type is nothing but building class by inheriting properties and behavior of Exception class.
Every error class must inherit Exception class.

Basic steps for developing custom error type or user defined error type

1. Define class by inheriting Exception class
2. Include pass statement (OR) override __str__ method

Example:

```
class ZeroMultiplyError(Exception):
    pass
def multiply(n1,n2):
    if n1==0 or n2==0:
        raise ZeroMultiplyError()
    else:
        return n1*n2


while True:
    try:
        a=int(input("Enter first number "))
        b=int(input("Enter second number "))
        c=multiply(a,b)
        print(f'Product of {a}x{b}={c}')
        break
    except ValueError:
        print("input value must be integer type")
    except ZeroMultiplyError:
        print("cannot multiply number with zero")
```

Output

```
Enter first number 6
Enter second number 0
cannot multiply number with zero
Enter first number 4
```

```
Enter second number abc
input value must be integer type
Enter first number 4
Enter second number 5
Product of 4x5=20
```

Example:

```
users={'nit':'n123',
       'naresh':'n456',
       'ramesh':'ram321'}

class LoginError(Exception):
    def __str__(self):
        return "Invalid username or password"

def login(user,pwd):
    if user in users and users[user]==pwd:
        print(f'{user},Welcome')
    else:
        raise LoginError()


# Main
while True:
    try:
        print("******Login*****")
        print()
        uname=input("UserName :")
        password=input("Password :")
        login(uname,password)
        break
    except LoginError as a:
        print(a)
```

Output

```
******Login*****

UserName :nit
Password :abc
Invalid username or password
```

******Login*****

UserName :nit
Password :n123
nit,Welcome

Example:

```
class InsuffBalError(Exception):
    def __str__(self):
        return 'Insuff Funds or Balance'
class Account:
    def __init__(self,a,cn,b):
        self.__accno=a
        self.__cname=cn
        self.__balance=b
    def deposit(self,a):
        self.__balance+=a
    def withdraw(self,a):
        if self.__balance<a:
            raise InsuffBalError()
        else:
            self.__balance-=a
    def __str__(self):
        return f'{self.__accno},{self.__cname},{self.__balance}'


acc1=Account(101,"naresh",54000)
print(acc1)
try:
    acc1.deposit(1000)
    print(acc1)
    acc1.withdraw(2000)
    print(acc1)
    acc1.withdraw(90000)
    print(acc1)
except InsuffBalError as a:
    print(a)
```

Output

101,naresh,54000

101,naresh,55000
101,naresh,53000
Insuff Funds or Balance

Assert keyword (Assertion)
An assertion in Python is a statement used to test if a condition is true. If the condition is true, the program continues to execute normally. However, if the condition is false, an AssertionError is raised, and the program stops.

Purpose:
Debugging:
Assertions help identify bugs early in the development process by verifying assumptions about the program's state.

assert condition, message

condition: A boolean expression that is evaluated.
message: An optional string that is displayed if the condition is false.

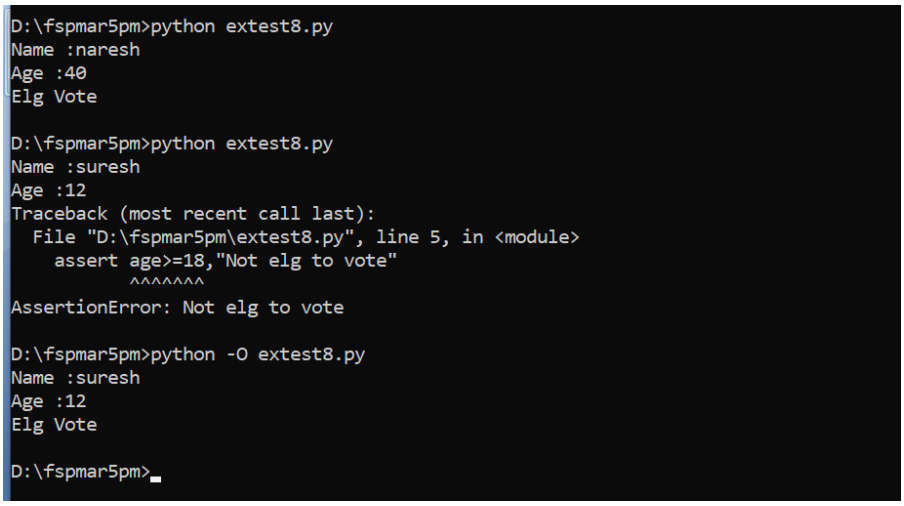
How to disable assertion?
-O : remove assert and __debug__-dependent statements;

python –o program-name

Example:
```
# voter elg

name=input("Name :")
age=int(input("Age :"))
assert age>=18,"Not elg to vote"
print("Elg Vote")
```

Output


D:\fspmar5pm>python extest8.py
Name :naresh
Age :40
Elg Vote
D:\fspmar5pm>python extest8.py
Name :suresh
Age :12
Traceback (most recent call last):
File "D:\fspmar5pm\extest8.py", line 5, in <module>
assert age>=18,"Not elg to vote"
^^^^^^^
AssertionError: Not elg to vote
D:\fspmar5pm>python -O extest8.py
Name :suresh
Age :12
Elg Vote
D:\fspmar5pm>