```
def division(n1,n2):
    try:
        return n1/n2
    except ZeroDivisionError:
        return 0
    finally:
        print("inside finally block")


res1=division(5,2)
print(res1)
```

raise keyword

What is generating exception or error?
Generating exception or error is nothing but creating object of error class and giving to PVM (Python Virtual Machine).

How a function generates error or exception?
A function generates exception or error using "raise" keyword.

Syntax: raise <error-type>()

Advantage of exception handling or error handling is separating business logic and error logic.

Syntax:

```
def <function-name>([parameters]):
        statement-1
        raise error-type()
```

Example:

```
def multiply(n1,n2):
    if n1==0 or n2==0:
        raise ValueError()
```

```
    else:
        return n1*n2


try:
    num1=int(input("Enter first integer "))
    num2=int(input("Enter second integer "))
    num3=multiply(num1,num2)
    print(f'{num1}*{num2}={num3}')
except ValueError:
    print("cannot multiply numbers with zero")
```

Output

```
Enter first integer 8
Enter second integer 0
cannot multiply numbers with zero
```

Custom Error Types or User defined Error Types

Each error is a class or data type.
All error classes are inherited from Exception class.

Syntax:

```
class <error-class-name>(Exception):
        variables
        methods
```

Example:

```
class ZeroMultiplyError(Exception):
    def __init__(self):
        super().__init__()
        self.__message="Cannot Multiply Number with Zero"
    def __str__(self):
        return self.__message

def multiply(n1,n2):
    if n1==0 or n2==0:
```

```
        raise ZeroMultiplyError()
    else:
        return n1*n2


try:
    num1=int(input("Enter first integer "))
    num2=int(input("Enter second integer "))
    num3=multiply(num1,num2)
    print(f'{num1}*{num2}={num3}')
except ZeroMultiplyError as a:
    print(a)
except ValueError as b:
    print("input value must be integer")
```

Output

```
Enter first integer 5
Enter second integer 2
5*2=10

Enter first integer 8
Enter second integer 0
Cannot Multiply Number with Zero

Enter first integer 5
Enter second integer abc
input value must be integer
```

Example:

```
users={"naresh":"n123",
       "suresh":"s321",
       "kishore":"k999"}
class InvalidPasswordError(Exception):
    pass
class InvalidUserError(Exception):
    pass
```

```
def login(u,p):
    if u in users:
        if users[u]==p:
            print(f'{u} welcome')
        else:
            raise InvalidPasswordError()
    else:
        raise InvalidUserError()


print("*****Login Application*****")
while True:
    try:
        user=input("UserName :")
        pwd=input("Password :")
        login(user,pwd)
        break
    except (InvalidUserError,InvalidPasswordError):
        print("Invalid username or password")
```

Output

```
*****Login Application*****
UserName :naresh
Password :n321
Invalid username or password
UserName :kishore
Password :k999
kishore welcome
```

nested try blocks

try block within try block is nested try blocks.
Nested error handling is done using nested try blocks.

Syntax:

```
try:  -> outer try block
   statement-1
   statement-2
   try:  -> inner try block
      statement-3
      statement-4
  except <error-type>: → inner except block
     statement-5
except <error-type>: → outer except block
     statement-6
```

if there is an error inside outer try block, it is handled by outer except block.
If there is an error inside inner try block, it is handled by inner except block. If inner except block not able to handle error, it is given to outer except block.

Example:

```
try:
   print("inside outer try block")
   a=int(input("Enter first integer value"))
   try:
      print("inside inner try block")
      b=int(input("Enter second integer value "))
      c=a/b
      print(c)
   except ZeroDivisionError:
      print("cannot divide number with zero")
except ValueError:
   print("Input value must be integer")
```

Output

```
inside outer try block
Enter first integer value5
inside inner try block
```

```
Enter second integer value 0
cannot divide number with zero

inside outer try block
Enter first integer value6
inside inner try block
Enter second integer value abc
Input value must be integer
```

try .. else

Syntax-1:	**Syntax-2:**

try:
  statement-1
  statement-2
except <error-type>:
  statement-3
else:
  statement-4
	try:
	  statement-1
	  statement-2
	except <error-type>:
	  statement-3
	finally:
	  statement-4
	else:
	  statement-5

Example:

```
try:
    print("Inside try block")
    a=int(input("enter any integer value"))
    print(a)
except ValueError:
    print("inside except block")
else:
    print("inside else block")
finally:
    print("inside finally block")
```

Output

```
Inside try block
```

enter any integer valueabc
inside except block
inside finally block

The statements which have to be executed after execution of try block is included within else block. Else block is not executed if try block generates error.

Assertions

What are assertions?
assertion is condition given within program.
This condition can be enabled or disabled during execution of program.
Assertions are used for debugging purpose.

How to create assertion?
Assertion is created using a keyword "assert".

Syntax-1: assert <condition>
Syntax-2: assert <condition>:message