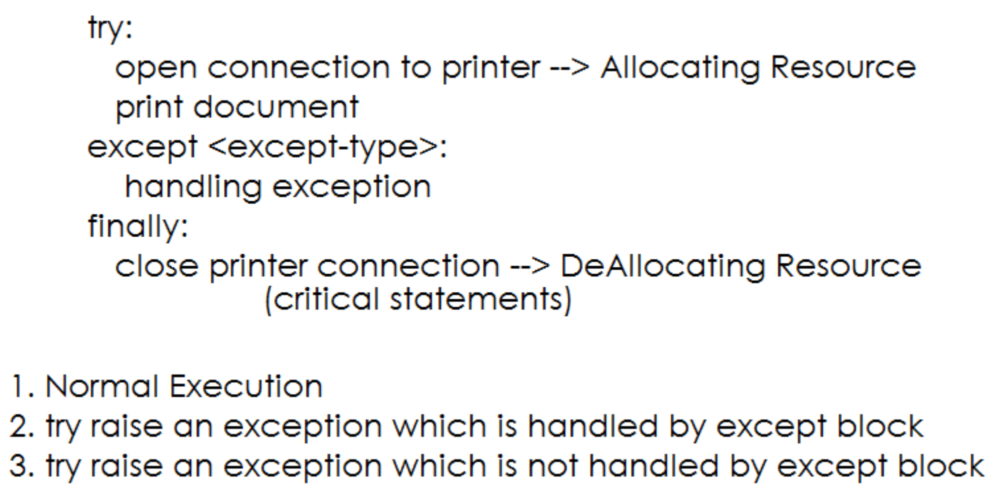
finally block
finally is not exception handler, it is block which is executed after execution of try block and except block.
Finally block is used to de-allocate resources allocated within try block.

try:
  statement-1
except <exception>:
  statement-2
finally:
  statement-3
	try:
	  statement-1
	finally:
	  statement-2
		try:
		  statement-1
		except <exception-1>:
		  statement-2
		except <exception-2>:
		  statement-3
		finally:
		  statement-4



```
try:
  open connection to printer --> Allocating Resource
  print document
except <except-type>:
   handling exception
finally:
  close printer connection --> DeAllocating Resource
                (critical statements)
```

1. Normal Execution
2. try raise an exception which is handled by except block
3. try raise an exception which is not handled by except block

Example:

```
def main():
    try:
        print("inside try block")
        n1=int(input("enter any number"))
        n2=int(input("enter any number"))
        n3=n1/n2
        print(n3)
    except ZeroDivisionError:
        print("inside except block")
        print("cannot divide number with zero")
    finally:
        print("inside finally block")

    print("continue...")
main()
```

Output:

```
========= RESTART: F:/python6pmaug/extest3.py ========
inside try block
enter any number5
enter any number2
2.5
inside finally block
continue...

========= RESTART: F:/python6pmaug/extest3.py ========
inside try block
enter any number6
enter any number0
inside except block
cannot divide number with zero
inside finally block
continue...
>>>
========= RESTART: F:/python6pmaug/extest3.py ========
inside try block
enter any number5
enter any numberab
inside finally block
Traceback (most recent call last):
  File "F:/python6pmaug/extest3.py", line 15, in <module>
    main()
  File "F:/python6pmaug/extest3.py", line 5, in main
    n2=int(input("enter any number"))
ValueError: invalid literal for int() with base 10: 'ab'
```

Example:

```
def main():
    try:
        f=open("file1.txt","r") # resource allocation
        s=f.read()
        print(s)
    except FileNotFoundError:
        print("Invalid File Name")
    finally:
        f.close() # resource deallocation
        print("file is closed")

main()
```

Output:

```
========= RESTART: F:/python6pmaug/extest4.py =========
pythonjavaoracle python is general purpose
    programming langauge101
file is closed
```

Advantage of exception handling is separate business and error logic.

raise keyword
this keyword is used to generate exception or error explicitly.

raising an exception or error is nothing but creating exception object and giving to python virtual machine.

Syntax: raise exception-class-name()

Example:

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

def main():
    num1=int(input("Enter First Number"))
    num2=int(input("Enter Second Number"))
    try:
        num3=multiply(num1,num2)
        print(num1,num2,num3)
    except ValueError:
        print("Cannot multiply number with zero")

main()
```

Output:

```
========= RESTART: F:/python6pmaug/extest5.py =========
Enter First Number5
Enter Second Number0
Cannot multiply number with zero
```

Creating user defined error classes or exception types or custom exceptions

Every exception or error is one class or data type. All these classes are inherited from exception class. Exception is a base class used to create exception types or error types.

Exception

All built-in, non-system-exiting exceptions are derived from this class. All user-defined exceptions should also be derived from this class.

Syntax:

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

Example:

```
class ZeroMultiplyError(Exception):
    def __init__(self):
        super().__init__()
    def __str__(self):
        return "Cannot Multiply Number withZero"
def multiply(n1,n2):
    if n1==0 or n2==0:
        raise ZeroMultiplyError()
    else:
        return n1*n2

def main():
    try:
        num1=int(input("Enter First Number"))
        num2=int(input("Enter Second Number"))
        num3=multiply(num1,num2)
        print(num1,num2,num3)
    except ValueError:
        print("Value must be integer")
    except ZeroMultiplyError as a:
        print(a)

main()
```

Output:

```
========= RESTART: F:/python6pmaug/extest5.py =========
Enter First Number5
Enter Second Number2
5 2 10
>>>
========= RESTART: F:/python6pmaug/extest5.py =========
Enter First Number5
Enter Second Number0
Cannot Multiply Number withZero
>>>
========= RESTART: F:/python6pmaug/extest5.py =========
Enter First Number5
Enter Second Numberab
Value must be integer
```

Example:

```
class InsuffBalError(Exception):
    def __init__(self):
        super().__init__()
    def __str__(self):
        return "Balance Not Available"
class Account:
    def __init__(self,a,c,b):
        self.__accno=a
        self.__cname=c
        self.__balance=b
    def deposit(self,a):
        self.__balance=self.__balance+a
    def withdraw(self,a):
        if a>self.__balance:
            raise InsuffBalError()
        else:
            self.__balance=self.__balance-a
    def __str__(self):
        return f'{self.__accno},{self.__cname},{self.__balance}'

def main():
    try:
        acc1=Account(101,"naresh",50000)
        print(acc1)
        acc1.deposit(10000)
        print(acc1)
        acc1.withdraw(90000)
        print(acc1)
    except InsuffBalError as i:
        print(i)
main()
```

Output:

```
========= RESTART: F:/python6pmaug/extest5.py =========
Enter First Number5
Enter Second Numberab
Value must be integer

========= RESTART: F:/python6pmaug/extest6.py =========
101,naresh,60000
Balance Not Available

========= RESTART: F:/python6pmaug/extest6.py =========
101,naresh,50000
101,naresh,60000
Balance Not Available
```

Example:

```
users_dict={'naresh':'nit123',
            'ramesh':'r123',
            'suresh':'k1234'}
```

```
class LoginError(Exception):
    def __init__(self):
        super().__init__()
    def __str__(self):
        return "Invalid UserName or Password"
def login(user,pwd):
    if user in users_dict and users_dict[user]==pwd:
        print(f'{user} welcome')
    else:
        raise LoginError()
def main():
    try:
        uname=input("UserName :")
        password=input("Password :")
        login(uname,password)
    except LoginError as l:
        print(l)
main()
```

Output:

```
========= RESTART: F:/python6pmaug/extest7.py =========
UserName :naresh
Password :nit123
naresh welcome

========= RESTART: F:/python6pmaug/extest7.py =========
UserName :naresh
Password :nit
Invalid UserName or Password
```

OS Module

OS is a predefined module in python. This module provides predefined functions to communicate with operating system. OS module functions are operating system dependent.

OS module provides the following functions.

1. Creating directory → mkdir()
2. Changing directory → chdir()
3. Finding current working directory → getcwd()

4. Removing directory → rmdir()
5. Renaming file
6. Listing files
7. Examine the properties of file

Example:
# write a python program to create folder

```
import os
fname=input("Enter Folder Name")
os.mkdir(fname)
print("Folder Created")
```

Output:
```
========= RESTART: F:/python6pmaug/ostest1.py =========
Enter Folder Namefolder1
Folder Created
```

Example:
# write a python program to change current working directory

```
import os

print(os.getcwd())
os.chdir("f:\\python6pmaug\\folder1")
print(os.getcwd())
f=open("file1","w")
print("file created")
```

Output:
```
========= RESTART: F:/python6pmaug/ostest2.py =========
F:\python6pmaug
f:\python6pmaug\folder1
file created
```

Example:
# write a program to remove directory or folder

```
import os
```

```
dname=input("Directory Name or Folder Name")
try:
    os.rmdir(dname)
    print("Directory is removed")
except OSError:
    print("Directory is not empty")
```

Output:

```
========= RESTART: F:/python6pmaug/ostest3.py =========
Directory Name or Folder Namefolder1
Directory is not empty
```