Ineuron Python Screening Assignment

1) Create a function in python to read the text file and replace specific content of the file.

In [11]:
```
def replace(file_name,replace_str,replace_with):
    with open(file_name,'r') as f:
        data=f.read()
    with open(file_name,'w') as f:
        f.write(data.replace(replace_str,replace_with))

def read(file_name):
    with open(file_name,'r') as f:
        data=f.read()
        return(data)
file_name = 'example.txt'
print("Original text : ",read(file_name))

replace_str = 'placement'
replace_with = 'screening'
replace(file_name,replace_str,replace_with)
print("Replaced text : ",read(file_name))
```

```
Original text :  This is a placement assignment
Replaced text :  This is a screening assignment
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Demonstrate use of abstract class, multiple inheritance and decorator in python using examples.

Multiple inheritance in python

multiple inheritance is a concept in which one class inheriting multiple classes.

In [7]:
```
class Father:
    def genes_father(self):
        print('Father genes')

class Mother:
    def genes_mother(self):
        print('Mother genes')

class Son(Father,Mother):
    def Son_g(self):
        print("son")


s=Son()
s.Son_g()
s.genes_mother()
s.genes_father()
```

```
son
Mother genes
Father genes
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Abstract class in Python

We can use an abstract class to define a set of methods that can be used by any child class. Abstract class have declaration but dosent have implementation. The abstract class can be used as a template for other classes.

In [17]:
```
from abc import ABC,abstractmethod

class Programmer(ABC):
    @abstractmethod
    def Programming(self):
        pass

class Cpp(Programmer):
    def Programming(self):
        print("I code in c++")

class Python(Programmer):
    def Programming(self):
        print("I code in Python")

class Java(Programmer):
    def Programming(self):
        print("I code in Java")


cpp = Cpp()
cpp.Programming()

python = Python()
python.Programming()

java = Java()
java.Programming()
```

```
I code in c++
I code in Python
I code in Java
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Decorator in Python

Decorator is used to add aditional functionality to any object with out modifing its structure.

In [22]:
```
def decorator(fun):
    def additional_func():
        print("This is additional function")
        fun()
    return  additional_func

@decorator
def Actual_function():
    print("This is actual function")


Actual_function()
```

```
This is additional function
This is actual function
```

In the above example the decorator function contains additional function which we have to add to the actual function with out distrubing the structure of Actual_function.

In [ ]: