Python Screening Assignment

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

File name	example.txt
Origin file content	This is a placement assignment
Replace string	Placement should be replaced by screening.
Replaced file content	This is a screening assignment

Soln:

```
def replace():
    fn = "example.txt"s
    with open(fn, 'r+') as f:
        text = f.read()
        text = re.sub('Placement', 'Screening', text)
        f.seek(0)
        f.write(text)
        f.truncate()
```

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

Soln:

Abstract Class:

In Python, the Abstract classes comprises of their individual abstract properties with respect to the abstract method of the respective class, which is defined by the keyword '@abstractproperty'.

Both the abstract class as well as the concrete class can be contained in the Abstract class. By using an abstract class we can define a generalized structure of the methods without providing complete implementations of every method. Abstract methods that are defined in the abstract class generally don’t have the body, but it is possible to have abstract methods with implementations in the abstract class and if any subclass is getting derived from such abstract class needs to provide the implementation for such methods.

Example1.py:

```
from abc import ABC, abstractmethod

class Shape(ABC):

    def common(self):

        print("This is a concrete method")

    def area(self):

        pass

    def perimeter(self):

        pass

class Square(Shape):

    def __init__(self,side):

        self.__side=side
```

```
    def area(self):
        return self.__side*self.__side
    def perimeter(self):
        return 4*self.__side


class Rectangle(Shape):
    def __init__(self,length,breath):
        self.__length=length
class Rectangle(Shape):
    def __init__(self,length,breath):
        self.__length=length
        self.__breath=breath
    def area(self):
        return self.__length*self.__breath
    def perimeter(self):
        return 2*(self.__length+self.__breath)
```

```
S1=Square(4)

print(S1.common())

print(S1.area())

print(S1.perimeter())

R1=Rectangle(2,4)

print(R1.common())

print(R1.area())

print(R1.perimeter())
```

Multiple Inheritance:

Multiple Inheritance is a type of inheritance in which one class can inherit properties of more than one parent classes. Python allows a class to inherit from multiple classes. If a class inherits from two or more classes, you'll have multiple inheritance.

To extend multiple classes, you specify the parent classes inside the parentheses () after the class name of the child class:

```
class ChildClass(ParentClass1, ParentClass2):
  pass
```

Example2.py:

```
class Car:
   def start(self):
      print('Start the Car')
   def go(self):
      print('Going')
```

```
class Can_Fly:
    def start(self):
        print('Start the Flyable object')
    def fly(self):
        print('Flying')


class FlyingCar(Can_Fly, Car):
    def start(self):
        super().start()


if __name__ == '__main__':
    car = FlyingCar()
    car.start()
```

Decorator:

Python has an interesting feature called decorators to add functionality to an existing code. A decorator in Python is a function that takes another function as its argument, and returns yet another function. Decorators can be extremely useful as they allow the extension of an existing function, without any modification to the original function code.

To create a decorator function in Python, we can create an outer function that takes a function as an argument. There is also an inner function that wraps around the decorated function.

Python has an interesting feature called decorators to add functionality to an existing code.

Example3.py:

```
def divide(x,y):
    print(x/y)
def outer_div(func):
    def inner(x,y):
        if(x<y):
            x,y = y,x
```

```
        return func(x,y)
    return inner
divide1 = outer_div(divide)
divide1(2,4)
```