CHANNABASAV ANGADI

**Mail : -** [**channulaikat@gmail.com**](mailto:channulaikat@gmail.com)

**GitHub :- https://github.com/ChannabasavAngadi**

**Phone : - 9343406062**

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

# Python program to replace text in a file  
x = input("enter text to be replaced:")  
y = input("enter text that will replace:")  
  
# file.txt should be replaced with  
# the actual text file name  
f = open("example.txt", "r+")  
  
# each sentence becomes an element in the list l  
l = f.readlines()  
  
# acts as a counter to know the  
# index of the element to be replaced  
c = 0  
for i in l:  
 if x in i:  
  
 # Replacement carries the value  
 # of the text to be replaced  
 Replacement = i.replace(x, y)  
  
 # changes are made in the list  
 l = Replacement  
 c += 1  
  
# The pre existing text in the file is erased  
f.truncate(0)  
  
# the modified list is written into  
# the file thereby replacing the old text  
f.writelines(l)  
f.close()  
print("Text successfully replaced")

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

* **An abstract class is a class, but not one you can create objects from directly. Its purpose is to define how other classes should look like, i.e. what methods and properties they are expected to have.**
* **The methods and properties defined (but not implemented) in an abstract class are called abstract methods and abstract properties. All abstract methods and properties need to be implemented in a child class in order to be able to create objects from it.**
* **We can create an abstract class by inheriting from the ABC class which is part of the** [**abc module**](https://docs.python.org/3/library/abc.html)**.**

# Python program showing  
# abstract base class work  
  
from abc import ABC, abstractmethod  
  
class AnimalKingdom(ABC):  
  
 @abstractmethod  
 def animal(self):  
 pass  
  
class Dog(AnimalKingdom):  
  
 # overriding abstract method  
 def animal(self):  
 print("I Bark")  
  
class Cat(AnimalKingdom):  
  
 # overriding abstract method  
 def animal(self):  
 print("I Meow")  
  
class Snake(AnimalKingdom):  
  
 # overriding abstract method  
 def animal(self):  
 print("I Hissssss")  
  
  
  
# Driver code  
R = Dog()  
R.animal()  
  
K = Cat()  
K.animal()  
  
R = Snake()  
R.animal()

**2.Multiple inheritance**

* **In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class. In this section of the tutorial, we will discuss inheritance in detail.**
* **In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.**

class Animal:  
 def speak(self):  
 print("Animal Speaking")  
 # child class Dog inherits the base class Animal  
  
  
class Dog(Animal):  
 def bark(self):  
 print("dog barking")  
  
  
class Cat(Dog):  
 def Meow(self):  
 print("Cat talking")  
  
# creting object for cat  
# cat inheriting dog  
# dog inheriting speak  
d = Cat()  
d.Meow()  
d.bark()  
d.speak()

**3.Decorator**

**Decorators are one of the most helpful and powerful tools of Python. These are used to modify the behavior of the function. Decorators provide the flexibility to wrap another function to expand the working of wrapped function, without permanently modifying it.**

def add(x):  
 return x + 1  
  
  
def sub(x):  
 return x - 1  
  
  
def operator(func, x):  
 temp = func(x)  
 return temp  
  
  
print(operator(sub, 10))  
print(operator(add, 20))