

**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 |

**ANSWER:**

Implementation done in the following github repository:

<https://github.com/aniketng21/INeuron_Assignments/tree/main/Python_Screening_Assignment>

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

**ANSWER:**

**DECORATORS:**

* Decorators are used to modify the behavior of the function.
* Decorators provide the flexibility to wrap another function to increase the usage of wrapped function, without permanently modifying it.
* In python, everything is an object.
* Functions in python are first class citizens, they have the capability to pass a function as an argument as well as return a function.\

In the below example, we will see the use of decorator by creating a calculate function.

def calculate(func):

def compute(x,y):

if(func.\_\_name\_\_ == "divide"):

print("Checking for Infinite Trap")

if y == 0:

print("You cannot divide with 0")

return

return func(x,y)

return compute

def divide(x,y):

return x/y

def sum(x,y):

return x+y

sum\_func = calculate(sum)

div\_func = calculate(divide)

sum\_result = sum\_func(5,0)

div\_result = div\_func(8,0)

print("RESULT: SUM=> ",sum\_result," DIVISION=> ",div\_result)

>>>

Checking for Infinite Trap

You cannot divide with 0

RESULT: SUM=> 5 DIVISION=> None

* In the above example, we are passing functions such as divide() and sum() to the calculate() function. The compute() function acts as a wrapper.
* Hence, we will be able to change the behavior of the function, according to the inner function.

We can use the above implementation in another syntactical implementation:

def calculate(func):

def compute(x,y):

if(func.\_\_name\_\_ == "divide"):

print("Checking for Infinite Trap")

if y == 0:

print("You cannot divide with 0")

return

return func(x,y)

return compute

@calculate

def divide(x,y):

return x/y

@calculate

def sum(x,y):

return x+y

sum\_result = sum(5,0)

div\_result = divide(8,0)

print("RESULT: SUM=> ",sum\_result," DIVISION=> ",div\_result)

>>>Checking for Infinite Trap

You cannot divide with 0

RESULT: SUM=> 5 DIVISION=> None

Here we use the @ symbol for decorator implementation.

**MULTIPLE INHERITANCE:**

* When a class can be derived from more than one base class this type of inheritance is called multiple inheritance.
* In multiple inheritance, all the features of the base classes are inherited into the derived class.

class Mother:

mothername = ""

def mother(self):

print(self.mothername)

class Father:

fathername = ""

def father(self):

print(self.fathername)

class Son(Mother, Father):

def parents(self):

print("Father :", self.fathername)

print("Mother :", self.mothername)

s1 = Son()

s1.fathername = "Narendra"

s1.mothername = "Anita"

s1.parents()

In the above example, the Son Class is inherited from the Mother and Father Class. Son Class has the ability to use all the methods available in both the classes. This is called multiple inheritance.

**ABSTRACT CLASS:**

* An abstract class allows you to create a set of methods that must be created within any child classes built from the abstract class.
* A class which contains one or more abstract methods is called an abstract class.
* By default, Python does not provide abstract classes.
* Python comes with a module that provides the base for defining Abstract Base classes(ABC) and that module name is ABC.
* A method becomes abstract when decorated with the keyword @abstractmethod.

EXAMPLE:

from abc import ABC,abstractmethod

class parent(ABC):

@abstractmethod

def name(self):

pass

class child(parent):

pass

child\_instance = child()

>>>

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

[<ipython-input-65-a4fbc4960a0f>](https://localhost:8080/#) in <module>()

**9** pass

**10**

---> 11 child\_instance = child()

TypeError: Can't instantiate abstract class child with abstract methods name

As we can see in the above example, we have created an abstract class parent. We have also defined an abstract method name().

Now we will create a child class which is inherited from the parent class. Using the abstract class, we have imposed a condition on the child class that it should have a method name() for it to get initiated.

In the above example, we have not created a method name() in the child class, hence we got the error.

Here is the appropriate implementation:

from abc import ABC,abstractmethod

class parent(ABC):

@abstractmethod

def name(self):

pass

class child(parent):

def name(self):

print("Aniket")

child\_instance = child()

child\_instance.name()

>>>Aniket