*Python Lab(Sem IV), Batch S13, 03/03/2022 Batch Prof. Dr. Shachi Natu*

## Abstract Classes

- Any class that is derived from class ABC which is present in module abc is called as Abstract Class
- ABC class is called Metaclass.
- Abstract class can have abstract methods and concrete methods.
- Abstract methods does not contain any function body in abstract class.
- Abstract methods are redefined in the classes derived from abstract class.
- Thus Abstract classes need to be extended and abstract methods need to be redefined in child class.
- If child class does not contain the action definition for abstract method, then it must be made abstract class
- Concrete methods have function body defined in abstract class.
- PVM can not create object of abstract class.
- abstract methods are declared using @abstractmethod decorator.
- If there is any abstract method in class, that class becomes abstract class.

In [1]:

```
from abc import ABC, abstractmethod

class sample(ABC):      #Abstract class

    @abstractmethod     #abstract method
    def disp(self):
        pass

    def show(self):    # concrete method
        print ("This is concrete method defined in abstract class")


```

In [2]:

```
myobject = sample()  #can not create object of abstract class
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-44f8b1c8a3c2> in <module>()
----> 1 myobject = sample()  #can not create object of abstract class

TypeError: Can't instantiate abstract class sample with abstract methods dis
p
```

In [3]:

```
class child(sample):

    def disp(self):
        print("This is definition of abstract method and is in child class")
```

In [4]:

```
c = child()
```

In [5]:

```
c.disp()
```

```
This is definition of abstract method and is in child class
```

In [6]:

```
c.show()
```

```
This is concrete method defined in abstract class
```

## Interface

- There is no explicit concept of interface available in Python like other languages.
- In python, Interface is an abstract class which contains all abstract methods and no concrete method.
- Being an abstract class, we can not create object of Interface.
- If a class is implementing an Interface, it must define all abstract methods from that interface.
- If a class does not implement all abstract methods from an Interface, that class becomes the abstarct class.
- Interface is used when all features from it are to be implemented differently for different objects.

In [7]:

```
from abc import ABC, abstractmethod

class sample(ABC):      #Abstract class

    @abstractmethod     #abstract method
    def disp(self):
        pass
    @abstractmethod
    def show(self):    # abstract method
        pass


```

In [16]:

```
#all abstract methods from interface are implemented
class child(sample):

    def disp(self):
        print("disp method from child class")

#     def show(self):
#         print("show method from child class")
```

In [17]:

```
1 c = child()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-cfb9ddeee44d> in <module>()
----> 1 c = child()

TypeError: Can't instantiate abstract class child with abstract methods show
```

In [10]:

```
1 c.disp()
```

```
disp method from child class
```

In [11]:

```
1 c.show()
```

```
show method from child class
```

In [12]:

```
1 #all abstract methods from interface are not implemented
2 class child1(child):
3
4     def disp(self):
5         print("disp method from child class")
6
7 #     def show(self):
8 #         pass
```

In [13]:

```
1 c = child1()
```

In [14]:

```
1 c.show()
```

```
show method from child class
```

In [15]:

```
1 c.disp()
```

```
disp method from child class
```

In [ ]:

```
1
```