Abstract Class and Interface :

- Class is a collection of non abstract methods
- Abstract collection of both abstract and non abstract methods
- Abstract method is a method which is not having any body or implementation and it should be defined by using predefined decorator "@abstractmethod"
- If you write any abstract method in the abstract class then the class should be defined as sub class of class "ABC" (Abstract Class)

```
Eg: @abstractmethod
def method1(self):
    pass
```

- non abstract method is having a body or implementation

```
Eg:  def method1(self):
         print("non abstract method")
```

- Null body method is not having any body or implementation , But null body method acts as non abstract method , But Every non abstract method is not an abstract method

```
Eg:  def method1(self):
         pass
```

"ABC" and "abstractmethods" are from "abc" module.

Syn for abstract class:

```
from abc import abstractmethod,ABC
class <ClassName>:
      fields
      non abstract methods
      abstract methods
```

Ex1: abstract class with abstract methods can't be instantiation

```
from abc import ABC,abstractmethod
class Sample(ABC):
   @abstractmethod
   def method1(self):
       pass

s=Sample()  #TypeError Abstract can't be instantiated
```

Note 2: abstract class without abstract method can be instantiated

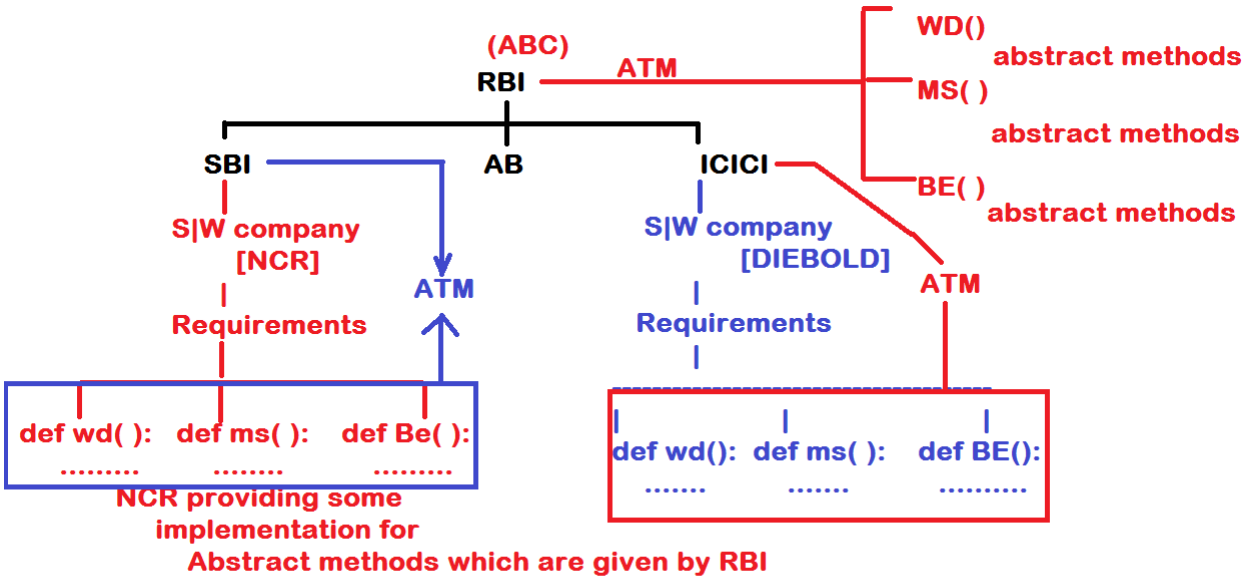
```
from abc import ABC,abstractmethod
class Sample(ABC):
   def method1(self):
       pass

s=Sample()  //valid
```

Note 3: if required we can also define abstract methods in a class but it is not good programming practice and that class can be instantiated

```
class Super:
   @abstractmethod
    def method(self):
         pass
```

s=Super() //valid

If any class is inherited by an abstract class with an abstract methods then the sub class is also acts as an abstract class thus it can't be instantiated




```
from abc import abstractmethod,ABC

class Demo(ABC):
   def method1(self):  #null body method | non abstract
      pass

   def method2(self):
      print("Hello m2 ") #non abstract method

   @abstractmethod
   def method3(self):
      pass
```

Eg 2:

Creating an object for an abstract class is nothing but creating an object for any of its concrete class

- A concrete class is a class which is overridden all abstract methods of its superclass
- Every concrete class is the sub class , But Every Subclass is not a concrete class

```
from abc import abstractmethod,ABC

class Demo(ABC):
   def method1(self):
      print("Hello m1 ") #non abstract method

   @abstractmethod
   def method2(self):
      pass

class Test(Demo):
   def method2(self):
      print("OR Method2 of Demo")


#d=Demo() TypeError
t=Test()
t.method1()
t.method2()
```

Eg 3:

When to use abstract methods and abstract classes ?

- Whenever two or more sub classes are required to fulfill the some role through different implementation .

```
from abc import abstractmethod,ABC

class Shapes(ABC):
   def __init__(self,dim1,dim2):
      self.dim1=dim1
      self.dim2=dim2
```

```
    @abstractmethod
    def findArea(self):
        pass


class Rect(Shapes):
    def findArea(self):
        return (self.dim1*self.dim2)

class Triangle(Shapes):
    def findArea(self):
        return (0.5*self.dim1*self.dim2)

r=Rect(4.0,4.0)
area_of_rect=r.findArea()
print("Area of Rect : ",area_of_rect)

r=Triangle(5.0,5.0)
area_of_tri=r.findArea()
print("Area of Triangle : ",area_of_tri)
```

Interface Example

- Class is the collection of non abstract methods
- Abstract class is the collection of both abstract or non abstract methods
- Interface is collection of abstract methods only
- Interface is also called pure abstract class

```
Syn:
from abc import abstractmethod,ABC
class ClassName:
    @abstractmethods
     def method(self):
         pass

    @abstractmethods
     def method2(self):
         pass
```

Note :

- IF any Class is inherited by an interface then all the abstract methods of the interface must be overridden otherwise it will be considered by python as an abstract class.

- Interface can't be instantiated. Creating an Object for an interface is nothing but creating an object for any of its implemented class.

```
from abc import abstractmethod,ABC

class Demo(ABC):
   @abstractmethod
   def method1(self):
      pass


class Sample(Demo):
   def method1(self):
      print("OR method1 of Demo")

d=Sample()
d.method1()
```

Example:

```
from abc import ABC,abstractmethod

class RBI(ABC):   #act as a an interface
   @abstractmethod
   def wd(self):
      pass
   @abstractmethod
   def ms(self):
      pass
   @abstractmethod
   def be(self):
      pass

class SBI(RBI):
   def wd(self):
      print("Wd Done By SBI")

   def be(self):
      print("BE Done By SBI ")
```

```
    def ms(self):
        print("MS Done By SBI")

atm=SBI()
atm.wd()
atm.be()
atm.ms()
```