In [ ]:
```
#Concept of Abstraction--> Hiding of irrelaveny data from the user. such that user can only
access the properties and behaviour of that functionality without knowing internal implementation
of that functionality
Example:
    Television
    ATM Machine
    Laptop
    Mobile
    Fan
    Web Applications
    Android Application
```

In [ ]:
```
#Abstract is something which does not talk about completiness . It is just partial
#implementation of anything.
```

In [ ]:
```
#Abstract method
--> Sometimes we don't know the implementation of a method still we need to declare
a  method such type of method are known as abstract method.(abstrct method have only declaration
                                                            not implementation)
-->In python if you want to declare abstract method then you need use @abstractmethod decorator.
Example":
    @abstractmethod
    def getnoofwheel():
        pass
--> @abstractmethod decorator is present in abc module. for declaring any method as abstrct you need
to import abc module(Abstract base class)
```

In [2]:
```
from abc import *
class test:
    @abstractmethod
    def m1(self):
        pass
Note: Child class is responsible to implement abstract method of parent class.
```

In [ ]:
```
#Abstract Class:
Sometimes we don't know the complete implementation of a class still we need to declare or define
a  classs such type of classes are known as abstract classes.
Every abstract class is a child class of  ABC class which is present in abc module.
In abstrct class it is mandatory that atleast one method should be abstract
We cannot create the object of abstract class.
```

In [ ]:
```
Case1:
    from abc import *
    class Test:
        pass
#It is not an abstract class. We can create the object of the above class
```

In [ ]:
```
case2:
    from abc import *
    class Test(ABC):
        pass
#In the above class we can create the object even it is derived from abc class because
it doesnot contain any abstract method
```

In [ ]:
```
case3:
    from abc import *
    class test(ABC):
        @abstractmethod
        def m1(self):
            pass
#We cannot create the object of it the reason is it is a child class of ABC and also having atleast
one abstrct method
```

In [3]:
```
case4:
from abc import *
class test(ABC):
    @abstractmethod
    def m1(self):
        pass
class child(test):
    @abstractmethod
    def m1(self):
        pass

c=child()
#Child is reponsible to implement abstrct method if child is not implementing
#it then the child class is also an abstract class so we cannot create the object of child as well as
#parent
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <cell line: 11>()
      7     @abstractmethod
      8     def m1(self):
      9         pass
---> 11 c=child()

TypeError: Can't instantiate abstract class child with abstract method m1
```

In [4]:
```
#case5:
from abc import *
class test(ABC):
    @abstractmethod
    def m1(self):
        pass
class child(test):
    pass
#If we are  not creating the object of above code then it is valid because
#synatactically it is correct but if you create an object then it is not possible.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [4], in <cell line: 10>()
      7 class child(test):
      8     pass
---> 10 c=child()

TypeError: Can't instantiate abstract class child with abstract method m1
```

In [11]:
```
#Example of Abstract class
from abc import *
class Vechile(ABC):
    @abstractmethod
    def getwheels(self):
        pass
    def Engine(self):
        return "230CC"
class Bus(Vechile):
    def getwheels(self):
        return 8
class Auto(Vechile):
    def getwheels(self):
        return 3
class Bike(Vechile):
    def getwheels(self):
        return 2
T=Bus()
T.Engine()
#Note Abstrct class can have abstrct methods as well as concrete(normal) method.
#In abstrct class you can access concrete method directly but you can access concrete method
#with the child class object
```

Out[11]:
```
'230CC'
```

In [ ]:
```
#Interface --> Interface concept is not in Python. But In java we have a syntax  for interfaces.
whereas in python we don't have something like this.

#What is an interface?
If any abstrct class is having all methods as abstract such type of abstrct class are known as
interfaces.
in python there is no any syntax for interface but we can achieve interface with the help
of abstract class and method
```

In [17]:
```
#Example of Interface
from abc import *
class DBInterface(ABC):
    @abstractmethod
    def connect(self):
        pass

    @abstractmethod
    def disconnect(self):
        pass

class Oracle(DBInterface):
    def connect(self):
        print("connecting to database")
    def disconnect(self):
        print("Disconnecting the database")

t=Oracle()
t.connect()
t.disconnect()

#Note: It is mandatory for child class to implement all abstrct method if child class is not
#implementing all abstrct method then we cannot create the object of child class
```

```
connecting to database
Disconnecting the database
```

In [ ]:
```
abc --> is a module --> if you want to create any abstrct method or class you need to import this
ABC --> is a Parent class --> each and every abstrct class must be a child class of ABC class.
```

In [ ]:
```
#Garbage Collection --> In old programming langauge like c/c++ we need to deleted the unsed variables
and data. programmer is reponsible to deleted or remove the irrelavent data.
In Python we have on assitant whose name is garbage collection and he will take care of useless thing
Garbage collection automatically delete the usless datat from the memeory
If the object doesnot have any reference variable then garbage collector will automcatically destroyed
that object.
```

In [22]:
```
#If we are wwriting any program internally garbage collector is running and he will destroy all the
#useless data  and reference variable
import gc
print(gc.isenabled())
gc.enable()
print(gc.isenabled())
```

```
False
True
```

In [ ]:
```
#Destructors
it is also a special method  and its name should be __del__.
Just before destrying the object garbage collector always calls destructors for performing the
Cleanup activies(closing database, relese memory)
Once Destructors completed clearnup activites then garbage collector destroy the object.
```

In [24]:
```
class test:
    def __init__(self):
        print("constructor is called")
    def __del__(self):
        print("Destructor is called")
t=test()
t=None
```

```
constructor is called
Destructor is called
Destructor is called
```

In [25]:
```
import sys
class Test:
    pass
t1=Test()
t2=t1
t3=t1
t4=t1
print(sys.getrefcount(t1))
```

```
5
```

In [ ]:
```
GetAttribute --> Get the data from the constrcutor
SetAttributes --> Set the data
```

In [ ]:
```
class  Student:
    def __init__(self,name,marks):
        self.__name=name
        self.__marks=marks
    def get_name(self):
        return self.__name
    def get_marks(self):
        return self.__marks
    def set_name(self,x):
        self.__name=x
    def set_marks(self,y):
        self.__name=y
```

In [28]:
```
x=isinstance(5,float)
x
```

Out[28]:
```
False
```

In [ ]:
```

```