In [1]:

```
#Inheritance in Python 3
class person:
    def method1(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
class Employee(person):
    def method2(self,e,f):
        self.e = e
        self.f = f
def main():
    x = person()
    x.method1(1,2,3)
    print(x.a,x.b,x.c)

    y = Employee()
    y.method1(4,5,6)
    y.method2(2,3)
    print(y.a,y.b,y.c)
    print(y.e,y.f)

if __name__ == "__main__":
    main()
```

```
1 2 3
4 5 6
2 3
```

In [2]:

```
#Polymorphism and Overriding
class person:
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
    def __str__(self):
        return self.a + " " + self.b + " " + self.c
class Employee(person):
    def __init__(self,a,b,c,d,e):
        self.d = d
        self.e = e
        super().__init__(a,b,c)
    def __str__(self):
        return super().__str__() + " "+ self.d + self.e
def main():
    x = person("debanik","roy",str(26))
    print(x)
    y = Employee("sirso","roy",str(32),"tall","cute")
    print(y)

if __name__=="__main__":
    main()
```

```
debanik roy 26
sirso roy 32 tallcute
```

In [3]:

```
#Polymorphism and Overriding
class X:
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
    def pt(self):
        return self.a,self.b,self.c
class Y(X):
    def __init__(self,a,b,c,d,e):
        self.d = d
        self.e = e
        super().__init__(a,b,c)
    def pt_1(self):
        return super().pt(),self.d,self.e

def main():
    x = X(1,2,3)
    print(x.pt())
    y= Y(1,2,3,4,5)
    print(y.pt_1())
if __name__=="__main__":
    main()
```

```
(1, 2, 3)
((1, 2, 3), 4, 5)
```

In [4]:

```
#Polymorphism with Hierarchical inheritance and Method Overriding
class animal:
    def pt(self):
        print("This are anumal")
    def char(self):
        print("need to do sex")
class girl(animal):
    def char(self):
        print("Do sex with boy")
class boy(animal):
    def char(self):
        print("Do sex with girl")

def main():
    x = animal()
    y = girl()
    z = boy()

    x.pt()
    x.char()

    y.pt()
    y.char()

    z.pt()
    z.char()
if __name__ == "__main__":
    main()
```

```
This are anumal
need to do sex
This are anumal
Do sex with boy
This are anumal
Do sex with girl
```

In [5]:

```
#Multuiple Inheritance and method regolution order and Hybrid Inheritance and Diamond of De

class A:
    def fun_A(self):
        print("fun_A")
class B:
    def fun_B(self):
        print("fun_B")
class C(A,B):
    def fun_C(self):
        print("fun_C")

def main():
    x = A()
    y = B()
    z = C()
    x.fun_A()
    y.fun_B()

    z.fun_C()
    z.fun_A()
    z.fun_B()
if __name__ == "__main__":
    main()
```

```
fun_A
fun_B
fun_C
fun_A
fun_B
```

In [6]:

```
#Hybrid Inheritance and Diamond of Death
class A:
    def fun_A(self):
        print("fun_A")
class B:
    def fun_B(self):
        print("fun_B")
class C(A,B):
    def fun_C(self):
        A.fun_A(self)
        B.fun_B(self)
        print("fun_C")

def main():
    x = A()
    y = B()
    z = C()
    x.fun_A()
    y.fun_B()

    z.fun_C()
if __name__ == "__main__":
    main()
```

```
fun_A
fun_B
fun_A
fun_B
fun_C
```

In [7]:

```
#Multilevel Inheritance and Hybrid Inheritance and Diamond of Death and method regolution o
class A:
    def main(self):
        print("fun_A")
class B(A):
    def main(self):
        print("fun_B")
class C(A):
    def fun_C(self):
        print("fun_C")
class D(B,C):
    def fun_D(self):
        print("fun_D")
        super().main()
def main():
    x = A()
    y = B()
    z = C()
    w = D()

    #x.main()

   # y.main()
    #
    #z.main()

    w.main()

if __name__ == "__main__":
    main()
```

```
fun_B
```

In [8]:

```
#Abstract Class and Abstract Method
from abc import ABC,abstractmethod
class X(ABC):
    @abstractmethod
    def x(self):
        print("class X")
class Y(X):
    def y(self):
        print("class y")
class Z(X):
    def y(self):
        print("class z")
def main():
    q = X()
    w = Y()
    e = Z()
    q.x()
    w.y()
    e.z()
if __name__ == "__main__":
    main()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-89ae623f3b2f> in <module>
     19     e.z()
     20 if __name__ == "__main__":
---> 21     main()

<ipython-input-8-89ae623f3b2f> in main()
     12         print("class z")
     13 def main():
---> 14     q = X()
     15     w = Y()
     16     e = Z()

TypeError: Can't instantiate abstract class X with abstract methods x
```

In [9]:

```
#Abstract Class and Abstract Method
from abc import ABC,abstractmethod
class X(ABC):              #private class
    @abstractmethod
    def x(self):
        print("class X")
class Y(X):
    def y(self):
        print("class y")
class Z():
    def z(self):
        print("class z")
def main():
    #q = X()
    #w = Y()
    e = Z()
    #q.x()
    #w.y()
    e.z()
if __name__ == "__main__":
    main()
```

```
class z
```

In [10]:

```
#Encapsulation, Access Modifiers, and Name Mangling
class A:
    def __init__(self):
        self.a = "Hey me Debanik"
        self._b = "Hey I am studing Robotics"
        self.__c = "AI is our future"

def main():
    q = A()
    print(q.a)
    print(q._b)
    print(q._A__c)
if __name__ == "__main__":
    main()
```

```
Hey me Debanik
Hey I am studing Robotics
AI is our future
```

In [ ]:

```

```