single inheritance ¶

In [18]:
```
class Person:
    def __init__(self,name):
        self.name=name
    def show(self):
        print(f"this is:{self.name}")
```

In [19]:
```
class Professor(Person):
    pass
```

In [20]:
```
p=Professor("xyz")
```

In [21]:
```
p.name
```

Out[21]:
```
'xyz'
```

In [22]:
```
p.show()
```

```
this is:xyz
```

In [23]:
```
class Human:
    def __init__(self,name):
        self.name=name
    def show(self):
        print(f"this is:{self.name}")
class Person(Human):
    pass
class Professor(Person):
    pass
```

In [24]:
```
p=Professor("D S SAVALIYA")
```

In [25]:
```
p.name
```

Out[25]:
```
'D S SAVALIYA'
```

In [26]:
```
p.show()
```

```
this is:D S SAVALIYA
```

multiple

In [27]:
```
class Person:
    def __init__(self,name):
        self.name=name
    def show(self):
        print(f"this is:{self.name}")
class techer:
    def credit(self):
        print("hii")
class python(Person,techer):
    pass
```

In [28]:
```
p=python("hello")
```

In [29]:
```
p.credit()
```

```
hii
```

In [30]:
```
p.name
```

Out[30]:
```
'hello'
```

In [31]:
```
p.show()
```

```
this is:hello
```

hierchical inheritance

In [32]:
```
class Human:
    def __init__(self,name):
        self.name=name
    def show(self):
        print(f"{self.name} is human")
class py(Human):
    pass
class java(Human):
    pass
```

In [33]:
```
p=py("python")
```

In [34]:
```
j=java("java")
```

In [35]:
```
p.name
```
Out[35]:
```
'python'
```

In [36]:
```
p.show()
```
```
python is human
```

In [37]:
```
j.name
```
Out[37]:
```
'java'
```

In [38]:
```
j.show()
```
```
java is human
```

constructor

In [39]:
```
class father:
    def __init__(self):
        self.money=1010000
        self.watch="ROLEX"
        print("father class")
    def show(self):
        print("father method")
class child(father):
    def __init__(self):
        self.money=777777
        self.car="RR"
        print("child")
    def show(self):
        print("son method")
```

In [40]:
```
c=child()
```
```
child
```

In [41]:
```
c.car
```
Out[41]:
```
'RR'
```

In [42]:
```
c.money
```
Out[42]:
```
777777
```

In [43]:
```
f=father()
```

```
father class
```

super method

In [44]:
```
class father:
    def __init__(self):
        self.money=1010000
        self.watch="ROLEX"
        print("father class")
    def show(self):
        print("father method")
class child(father):
    def __init__(self):
        super().__init__()
        self.money=777777
        self.car="RR"
        print("child")

    def show(self):
        super().show()
        print("son method")
```

In [45]:
```
c=child()
```

```
father class
child
```

In [46]:
```
c.watch
```

Out[46]:
```
'ROLEX'
```

In [47]:
```
c.car
```

Out[47]:
```
'RR'
```

In [48]:
```
c.show()
```

```
father method
son method
```

In [49]:
```
c.money
```

Out[49]:
```
777777
```

wap to a class book with attribute - title,author and pulisher . create a method 'diaplay()' to display the title and author . create a derive class from book with cource as attribute . the display() method of child class should override the parent class and display title author and course.

In [50]:
```
class book:
    def __init__(self,title,author,publisher):
        self.title=title
        self.author=author
        self.publisher=publisher
    def display(self):
        print(self.title,self.author,self.publisher)
class course(book):
    def __init__(self,course):
        super().__init__("python","mvp","lj")
        self.course=course
    def display(self):
        super().display()
        print(self.course)
```

In [51]:
```
c=course("py")
```

In [52]:
```
c.course
```

Out[52]:
```
'py'
```

In [53]:
```
c.author
```

Out[53]:
```
'mvp'
```

In [54]:
```
c.display()
```

```
python mvp lj
py
```

method overloading

In [68]:
```
class a :
    def car(self):
        print("bmw")
class b:
    def car(self):
        print("RR")
```

In [69]:
```
a1=a()
```

In [70]:
```
b=b()
```

In [71]:
```
a1.car()
```

```
bmw
```

In [72]:
```
b.car()
```

```
RR
```

abstract class

In [74]:
```
from abc import ABC,abstractmethod
from math import pi
```

In [82]:
```
class shape(ABC):
    @abstractmethod
    def area(self):
        pass
class rectangle(shape):
    def area(self,width,height):
        self.width=width
        self.height=height
        print(self.width*self.height)
class circle(shape):
    def area(self,r):
        self.r=r
        print(pi*self.r*self.r)
```

In [83]:
```
a=rectangle()
```

In [84]:
```python
a.area(10,20)
```
```
200
```

In [85]:
```python
c=circle()
```

In [81]:
```python
c.area(10)
```
```
314.1592653589793
```

In [ ]:
```python

```