Object Oriented Programming

BE(CSE) II-Semester

Prepared by
S. Durga Devi,
Assistant Professor,
CSE,CBIT

Unit-IV

Inheritance:

- Introduction
- Inheriting classes
- Polymorphism and method overloading
- Composition or Containerships
- Abstract classes and inheritance.

Operator Overloading:

- Introduction
- Implementation of Operator Overloading
- Overriding.

File Handling:

- File types, opening and closing files
- reading and writing files, file positions

Inheritance

- Is-A-Relation Vs Has-A- Relation
- Composition Vs Aggregation
- Boyfriend Vs Girl friend

  Boyfriend says : I cannot without you --- Strong Association

  Boyfriend Says: I can without you -----Weak Association

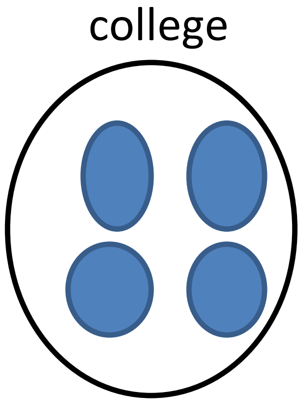
Here strong association means composition

weak association means Aggregation.

- College vs Department (composition)

  within college several departments are there like CSE,ECE etc.

  - without college is there department?
  - department associated with the college.
  - here department strongly associated with the college.
  - college is also called container object
  - departments are contained objects.
  - college has departments(Has is relation)



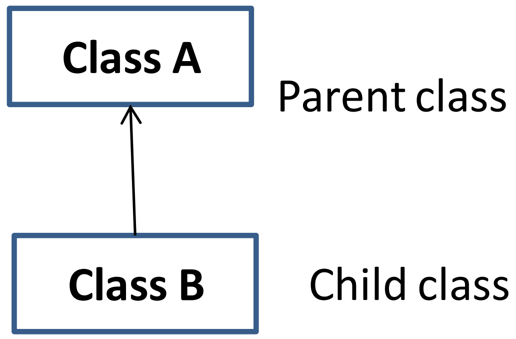
- Department has professors
- Suppose department has closed, the professors may moved to other departments . MCA department closed then professors moved to CSE/IT.
- Department and professors has a weak association- aggregation.

- Inheritance- IS-A-Relation
- Inheritance is a process of acquiring properties of other class is called inheritance.

- Advantage of inheritance is reusability.
- Extension of functionality

Inheritance in python

```
class  A: # parent class
         --- attributes
        ---- methods
  class B(A):  # class B extends class A
          --- attributes
          ----- methods
```



- Class that inherited from other class is called child class/derived class/sub class.
- Class from which other classes are derived is called parent class/base class/ super class.

Types of inheritance

1. Single inheritance
2. Multi level inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance
6. Cyclic inheritance

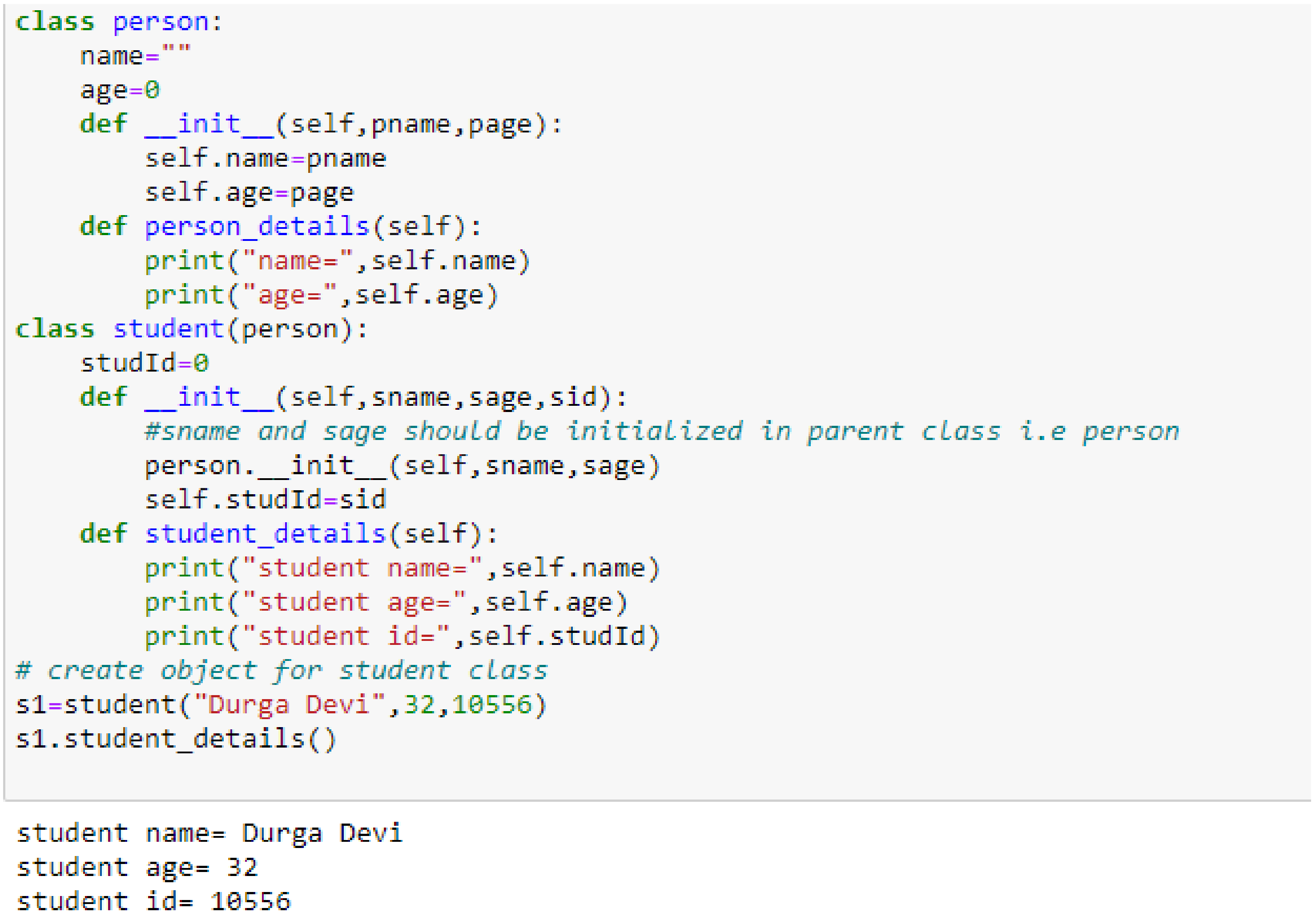
1.single inheritance

1. Single inheritance: Inherit from one parent to one child

```
class Parent:
            def m1(self):
                        print("parent class")
 class Child(Parent):
            def m2(self):
                        print("child class")
c=Child()
c.m1()
c.m2()
```

Note: Child class can have access to the Parent class methods and variables.

Example on inheritance

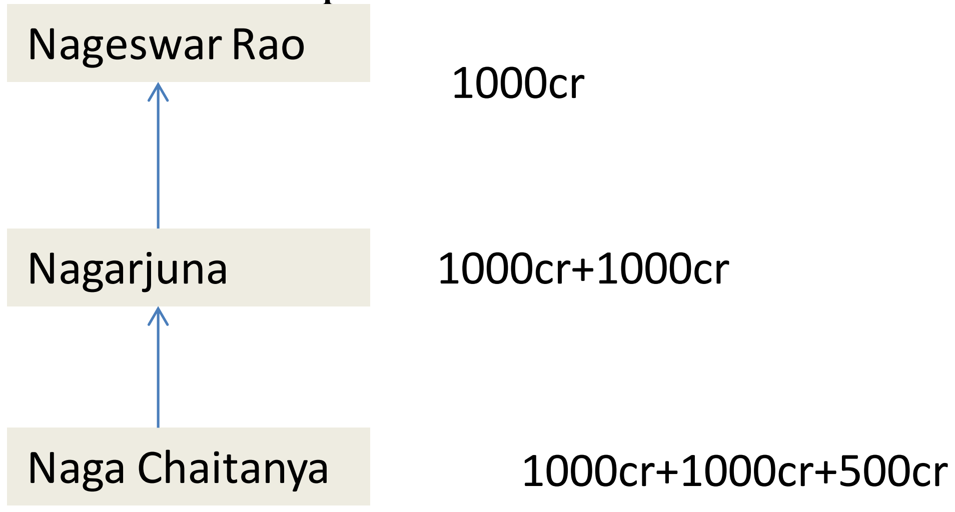


```
class person:
    name=""
    age=0
    def __init__(self,pname,page):
        self.name=pname
        self.age=page
    def person_details(self):
        print("name=",self.name)
        print("age=",self.age)
class student(person):
    studId=0
    def __init__(self,sname,sage,sid):
        #sname and sage should be initialized in parent class i.e person
        person.__init__(self,sname,sage)
        self.studId=sid
    def student_details(self):
        print("student name=",self.name)
        print("student age=",self.age)
        print("student id=",self.studId)
# create object for student class
s1=student("Durga Devi",32,10556)
s1.student_details()
```

```
student name= Durga Devi
student age= 32
student id= 10556
```

2. Multi level inheritance

- A class is derived from another class , in turn derived class acts as base class to another class. Derived class act as parent to other class



Multi level inheritance

Example on multilevel inheritance

```
class Nagaswarrao:
                def p1(self):
                                print("parent of Nagarjuna")
class Nagarjuna(Nagaswarrao):
                def p2(self):
                                print("parent of Naga Chaitanya")
class Nagachaitanya(Nagarjuna):
                def p3(self):
                                print("son of Nagarjuna")
x=Nagachaitanya()
x.p1()
x.p2()
x.p3()
```

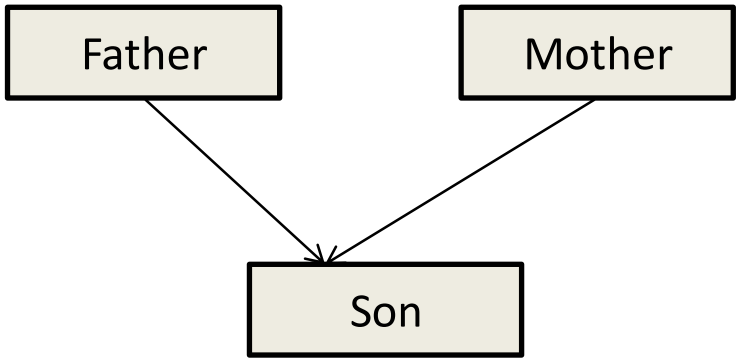
output

```
parent of Nagarjuna
parent of Naga Chaitanya
son of Nagarjuna
```

3. Multiple inheritance

Child class is inherited from more than one parent class is called multiple inheritance

- **Java does not support multiple inheritance**
- **Python supports multiple inheritance concept**



Syntax:

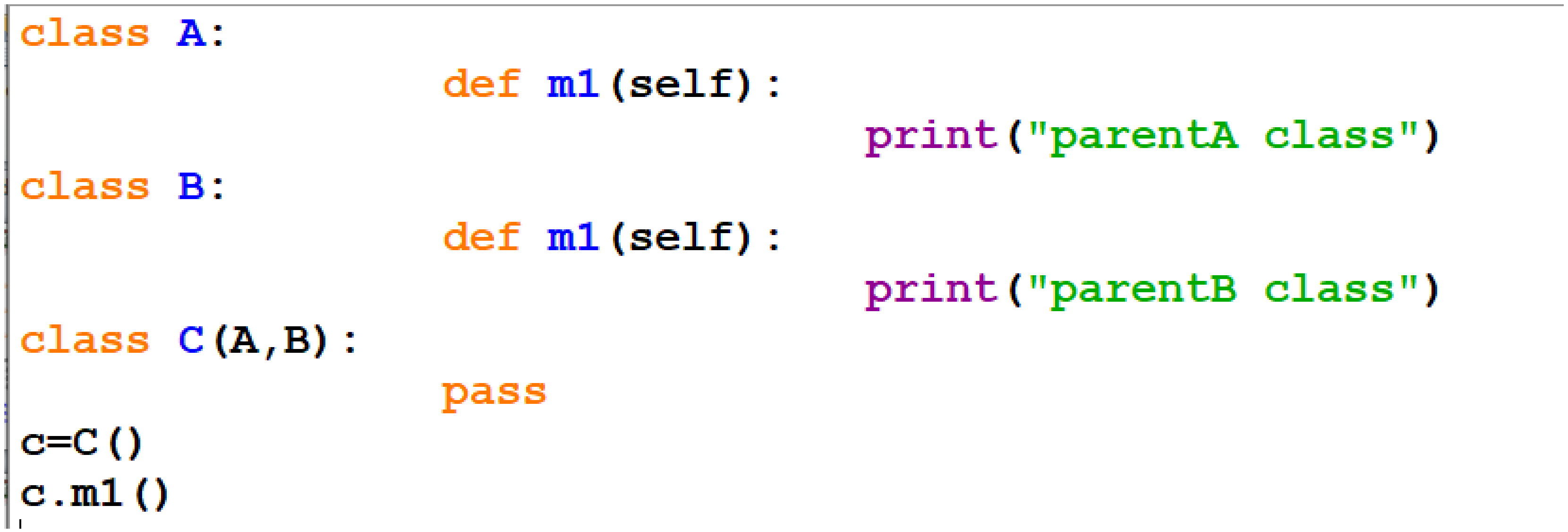
```
 class A:
         -----
 class B:
         ------
class C(A,B):
       ----
```

Example on Multiple inheritance

```
class A:
                def m1(self):
                                print("parentA class")
class B:
                def m2(self):
                                print("parentB class")
class C(A,B):
                pass
c=C()
c.m1()
c.m2()
```

```
parentA class
parentB class
```

What is the output of below program? Will you get any error ?



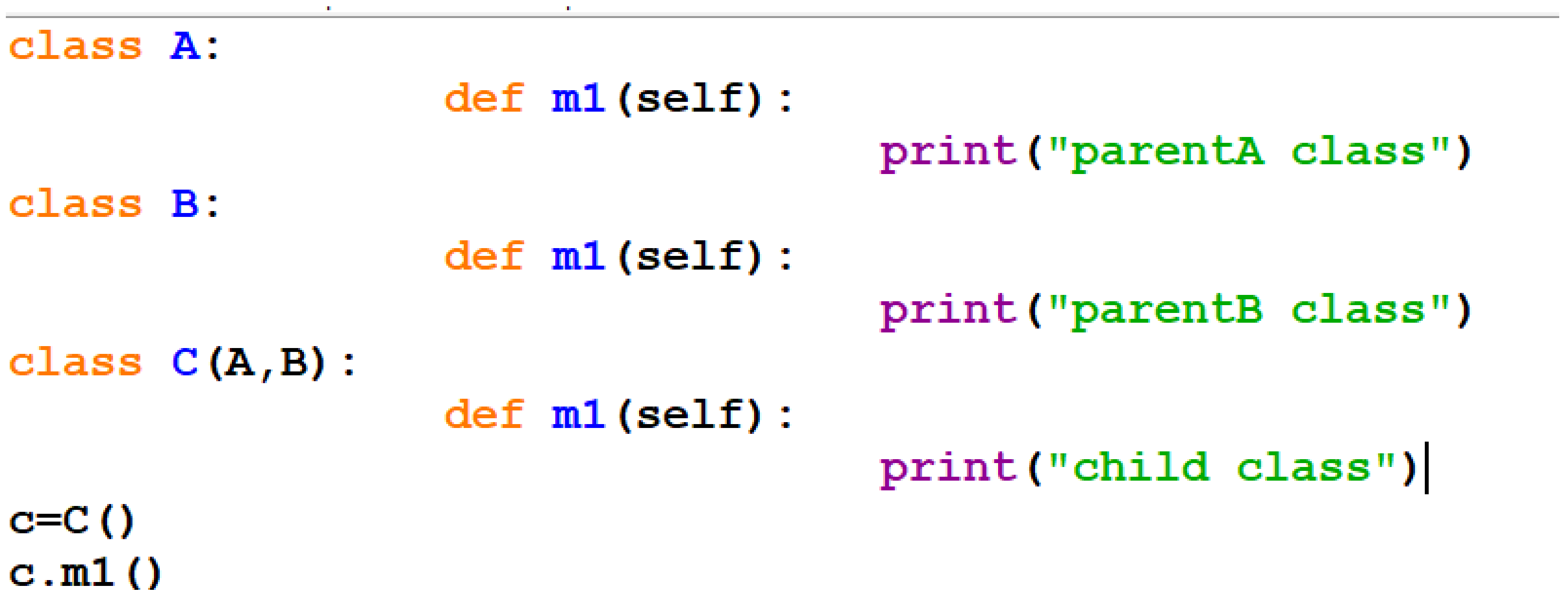
```
class A:
                def m1(self):
                                print("parentA class")
class B:
                def m1(self):
                                print("parentB class")
class C(A,B):
                pass
c=C()
c.m1()
```

Output is

parentA class

Note : output depends on order of parent classes defined in derived class.

What is the output of below program?



```
class A:
                def m1(self):
                                print("parentA class")
class B:
                def m1(self):
                                print("parentB class")
class C(A,B):
                def m1(self):
                                print("child class")
c=C()
c.m1()
```

Output

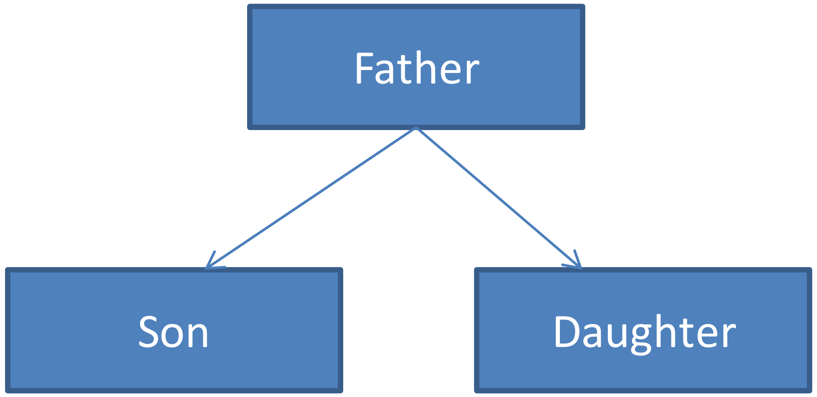
child class

Note: if child has the same method as parents class. Child class method will be executed.

4. Hierarchical inheritance

Single class parent to more than one child class.

- All child classes at the same level



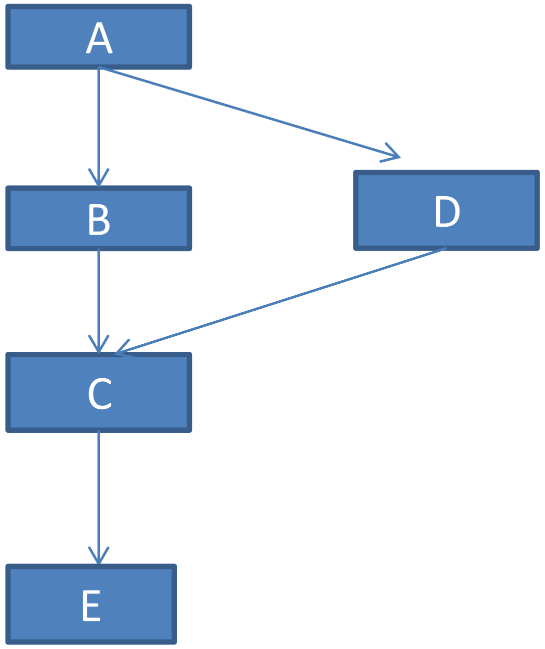
Example on Hierarchical inheritance

```
class Father:
        def m1(self):
                print("parent class")
class Son(Father):
        def m2(self):
                print("child1 class")
class Daughter(Father):
        def m3(self):
                print("child2 class")
d=Daughter()
d.m3() #child2 class
d.m1() #parent class
d.m2() #AttributeNameError
```

Note: though derived classes has same parent , attributes of one derived class cannot have access to attributes of another derived class.

5. Hybrid inheritance/multipath inheritance

- Mixing up of two or more types of inheritance is called inheritance



$$\begin{tikzcd}
A \arrow[d] \arrow[rd] & \\
B \arrow[d] & D \arrow[ld] \\
C \arrow[d] & \\
E &
\end{tikzcd}$$

S. Durga Devi ,CSE,CBIT

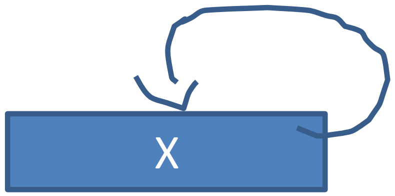
6. Cyclic inheritance

-A class extends by itself is called cyclic inheritance.

- Practically cyclic inheritance is not possible. Python does not support cyclic inheritance

example

```
class X(X):
```

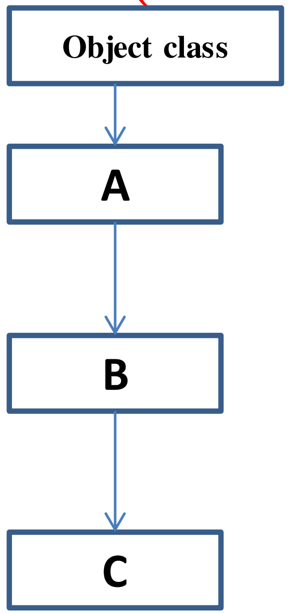


What is the need of using cyclic inheritance

- Suppose class X wants to access attributes of class Y and class Y also needs to access attributes of class Y
- class X(Y):
- class Y(X):

MRO(Method Resolution Order)

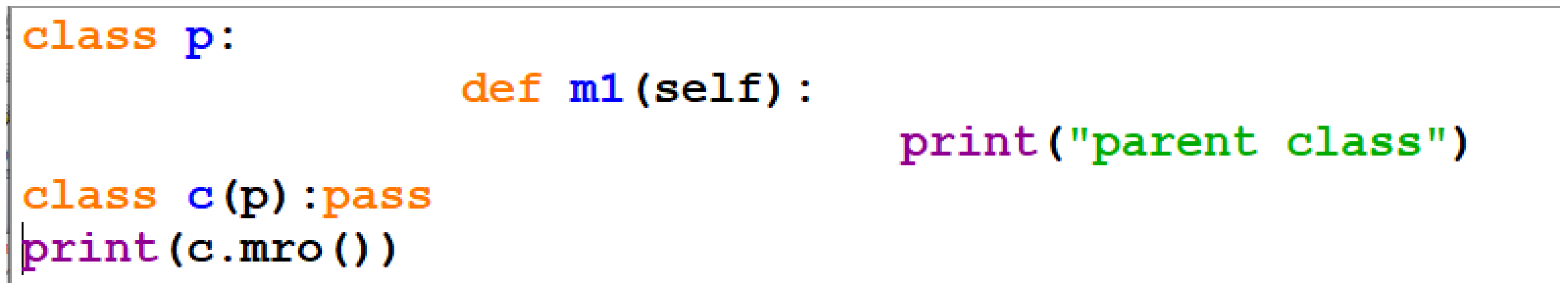
CASE-1



- In the above scenario, if you want to access any attribute first it will search in class C, if not will search in class B , still not in Class B it will search in class A, if even not in class A finally searched in Object class.

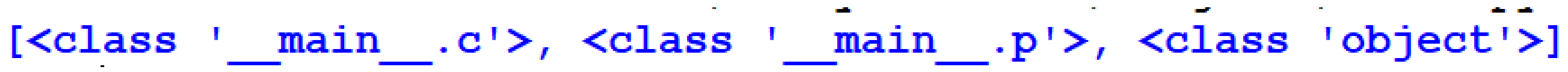
- The order of searching these classes is called Method Resolution Order.
- To know this method resolution order we can use predefined method called **mro().**

Example on mro()



```
class p:
                def m1(self):
                                print("parent class")
class c(p):pass
print(c.mro())
```

Output



```
[<class '__main__.c'>, <class '__main__.p'>, <class 'object'>]
```

Example on mro()

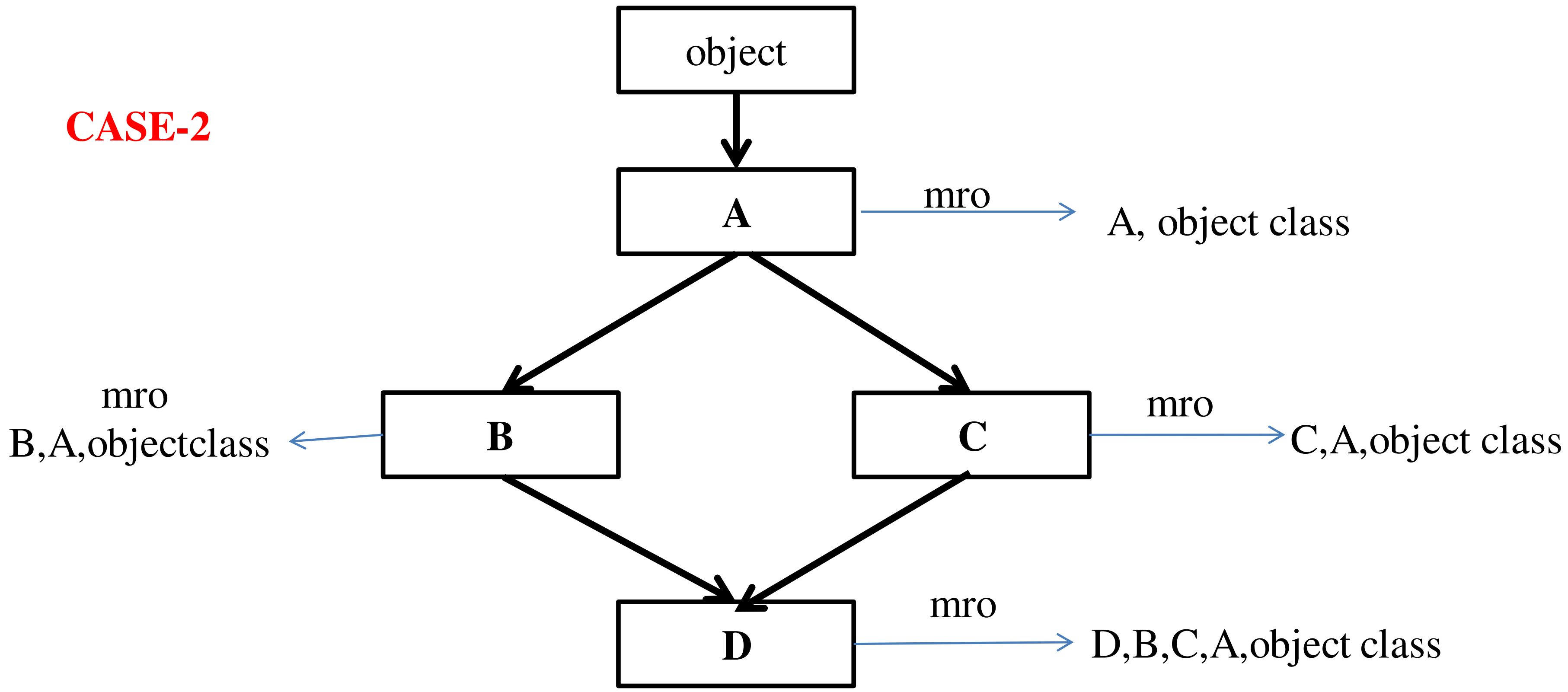
```
class p:
                def m1(self):
                                print("parent class")
class c(p):pass
x=p()
x.m1()
print(p.mro())
```

Output

```
parent class
[<class '__main__.p'>, <class 'object'>]
```

MRO

CASE-2

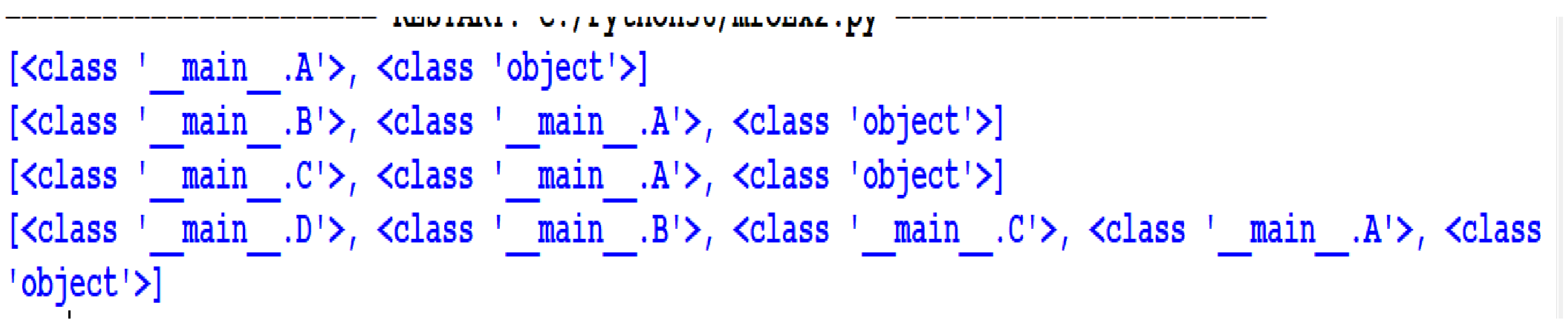


The problem in this case is also called as diamond problem.
- Method resolution order takes place from left to right.

Example for case 2(mro)

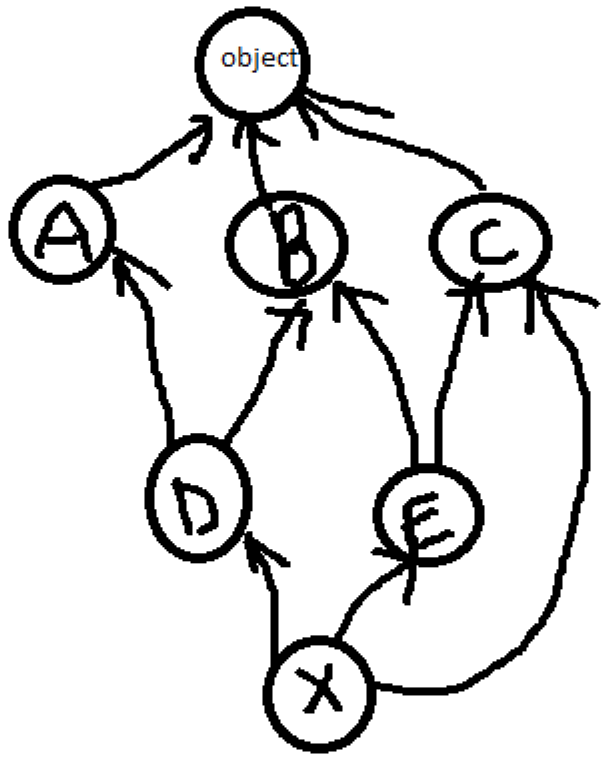
```
class A:pass
class B(A):pass
class C(A):pass
class D(B,C):pass
print(A.mro())
print(B.mro())
print(C.mro())
print(D.mro())
```

output



```
====================== RESTART: C:/Python36/mroEx2.py ======================
[<class '__main__.A'>, <class 'object'>]
[<class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
[<class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class
'object'>]
```

C3 algorithm

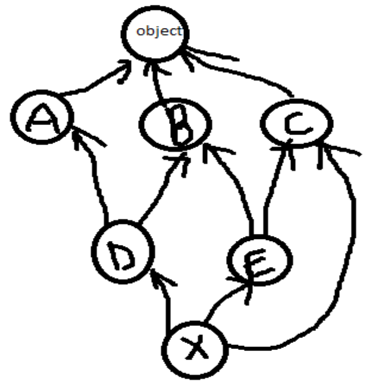


What is mro of class X?

➢In multiple levels with hybrid inheritance there is no guarantee of mro.
➢It uses the standard algorithm called C3 algorithm

mroEx3.py

C3 algorithm



- C3 algorithm
- Mro(X)=X+merge(mro(D),mro(E), mro(C), parent-list(X))
- mro(X)=X+merge(DABobject,EBCobject,Cobject,DEC)

Suppose take a list ABCDEFGH

First element ‘ A ‘ is a head element

‘BCDEFGH’ elements are called tail part

how do you merge

- **if the head element in first list not present in tail part of the other lists, then add head element in result and remove that element from the all the lists. Else take head element of second list.**

- Mro(X)=X+D+merge(ABobject,EBCobject,Cobject,EC)
- **Mro(X)=X+D+A+merge(Bobject,EBCojbect,Cobject,EC)**
- **Mro(X)=X+D+A+merge(Bobject,EBCobject,Cobject,EC) B present in other list then go to next list.**
- **Mro(X)=X+D+A+E+merge(Bobject,BCobject,Cobject,C)**
- **Mro(X)=X+D+A+E+B+merge(object,Cobject,Cobject,C)**
- **Mro(X)=X+D+A+E+B+C+merge(object,object,object)**
- **finally result is mro(X)=X,D,A,E,B,C,object**

Polymorphism and method overloading

- Poly means many and morph means forms
- Polymorphism means many forms which one of the main feature of OOPs.
- Inheritance is related to the classes whereas polymorphism is related to the methods.
- One name plays multiple roles.
- Yourself is a best example for the polymorphism. Suppose when you are at home you act as a son to your parents and brother to your sister, when you are in college you are a student, when you are outside of class act as a friend. Yourself is playing different roles in different context.

Polymorphism is implemented by overloading and overriding

Overloading

- Same operator or method is used for different purposes is called overloading.
- Ex1: + operator can be used for adding integer numbers as well as concatenate two strings

```
print(10+20) #30
print("Durga"+"Devi") #"DurgaDevi"
```

- Ex2: * operator can be used for multiplication and string repetation

```
print(2*3) # 6
print("CBIT"*3)  #CBITCBITCBIT
```

- In above two examples operators exhibit different behaviours is called operator overloading.
- Ex3: deposit(cash)

  deposit(cheque)

  deposit(DD)
- in example3 same method deposit with different arguments such as cash, cheque and DD.
- same method used for different purposes is called method overloading.

Overloading

- Three types of overloading

1. Operator overloading
2. Method overloading
3. Constructor overloading

1. Operator overloading

- Same operator used for different purposes is called operator overloading.
- Example to use + operator on student class object

```
class Student:
                    def __init__(self,name):

          self.name=name
  s1=Student("Durga")
  s2=Student("Devi")
 print(s1+s2)
```

Output

```
Traceback (most recent call last):
  File "F:/OOPPython-2020/Programs/opoverldEx.py", line 6, in <module>
    print(s1+s2)
TypeError: unsupported operand type(s) for +: 'student' and 'student'
```

Note: Here + operator cannot be extended to objects of a class to overcome this problem **operator overloading** is used

- Java programming does not supports the operator overloading
- python supports operator overloading .

- How python provides support for operator overloading?

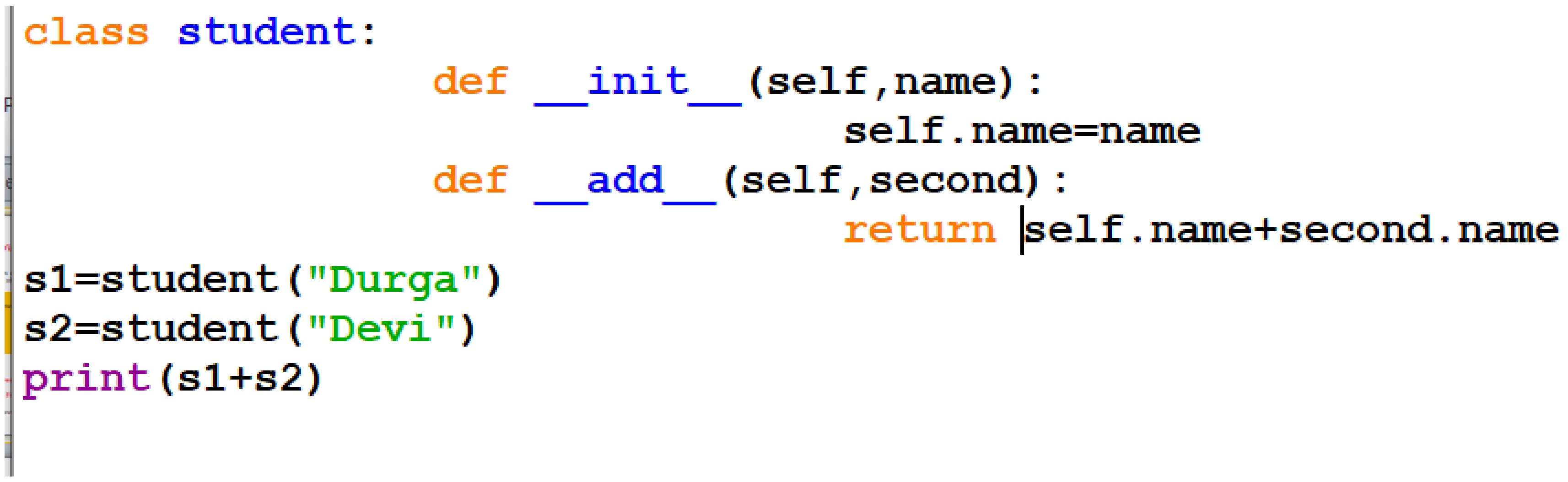
- Python internally defines a magic methods for every operator.
- When you want to overload an operator the corresponding magic method should be override in our class.

list of operators and their magic methods

Operator	Magic method	Operator	Magic method
+	__add__(self,x)	+=	__iadd__(self,x)
-	__sub__(self,x)	-=	__isub__(self,x)
*	__mul__(self,x)	*=	__imul__(self,x)
/	__truediv__(self,x)	/=	__idiv__(self,x)
**	__pow__(self,x)	>	__gr__(self,x)

For more details refer in prescribed text book

Program to overload + operator



```
class student:
                def __init__(self,name):
                                self.name=name
                def __add__(self,second):
                                return self.name+second.name
s1=student("Durga")
s2=student("Devi")
print(s1+s2)
```

Output

```
DurgaDevi
```

Case study:

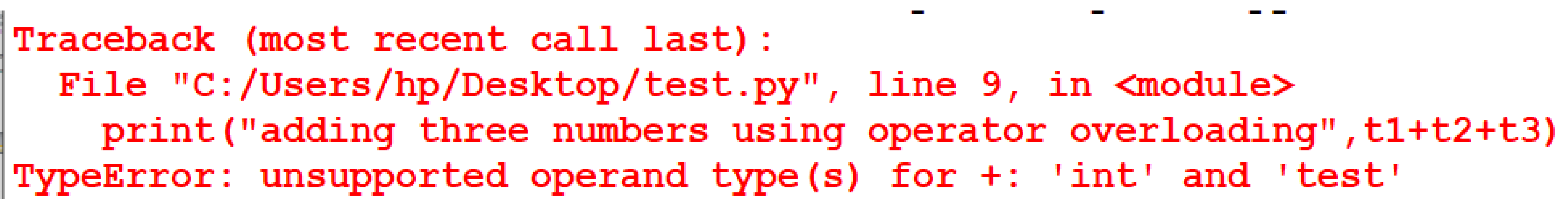
Calculate monthly salary of a particular employee by defining following classes.

1. Define a class ‘Employee’ which defines following methods.
   1. constructor with arguments name,salary
   2. magic method __mul__(self,x)

      __mul__() method overloads * operator to calculate employee salary based on the no of days he presented in particular month.
2. Define a class ‘employeeAttendance’ which defines constructor with argument is ‘days’

```
class test:
                def __init__(self,x):
                                self.x=x
                def __add__(self,second):
                                return self.x+second.x
t1=test(10)
t2=test(20)
t3=test(30)
print("adding three numbers using operator overloading",t1+t2+t3)
```

output



```
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/test.py", line 9, in <module>
    print("adding three numbers using operator overloading",t1+t2+t3)
TypeError: unsupported operand type(s) for +: 'int' and 'test'
```

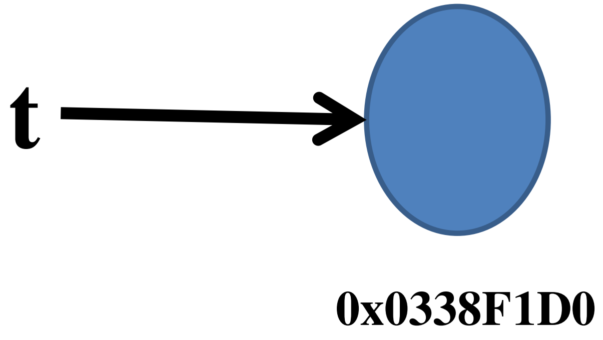
- In above example t1+t2 return type is int int we are trying to add to the t3(which is of type ‘test’). So int value cannot be added to the class object.

- To overcome this error use magic method __str__()

```
class test:
        def __init__(self):
                print("hai")
t=test()
print(t)
```

output

```
hai
<__main__.test object at 0x0338F1D0>
```



note: when you are trying to print object reference of a class internally python calls magic method called __str__() method

```python
class test:
        def __init__(self,x):
                self.x=x
        def __str__(self):
                return "sum is "+str(self.x)
        def __add__(self,second):
                total=self.x+second.x
                sum=test(total)
                return sum
t1=test(10)
t2=test(20)
t3=test(30)
print(t1+t2+t3)
```

```
sum is 60
```

2. Method overloading

- method overloading: when more than one method has same name but differ in their no of arguments and type of arguments is called method overloading.

- Ex: consider java code here

```
m1(int i)
m1(String s)
m1(float x,float y)
m1(int i, float j)
```

method m1() name is same but differ in their no of arguments and type of arguments

- **Python does not support the method overloading/constructor overloading**
- Why python does not support the method overloading?
- In python we are not declaring type of an argument, we can pass any type

-
  ```
  m1(i)
  m1(s)
  m1(x,y)
  m1(i,j)
  ```

- Really python does not require any method overloading.

Suppose python requires to use method overloading in some application.

How do you handle method overloading in python?

We can use method overloading in python indirectly by using any one of the below two ways.

1. Default arguments
2. Variable length arguments

- 1. method overloading with default arguments

Example:

```
class Test:
        def sum(self,a=None,b=None,c=None):
                if a!=None and b!= None and c!= None:
                        print('The Sum of 3 Numbers:',a+b+c)
                elif a!=None and b!= None:
                        print('The Sum of 2 Numbers:',a+b)
                else:
                        print('Please provide 2 or 3 arguments')

t=Test()
t.sum(10,20)
t.sum(10,20,30)
t.sum(10)
```

```
The Sum of 2 Numbers: 30
The Sum of 3 Numbers: 60
Please provide 2 or 3 arguments
```

2. Method overloading with variable length arguments

```
class Test:
                def sum(self,*a):
                                total=0
                                for x in a:
                                                total=total+x
                                print('The Sum:',total)
t=Test()
t.sum(10,20)       The Sum: 30
t.sum(10,20,30)    The Sum: 60
t.sum(10)          The Sum: 10
t.sum()            The Sum: 0
```

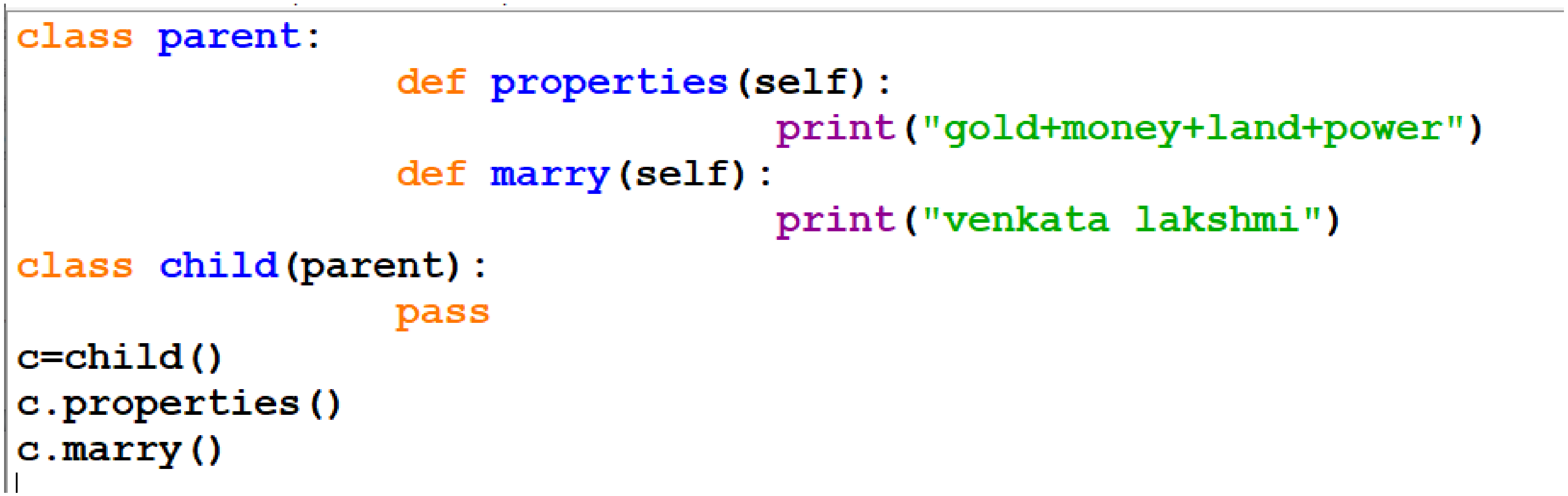
Overriding

- Overriding: same method name with different implementations defined in parent class as well as child class is called method overriding.
- As you know, what ever public members in parent class can have access to the child class. Suppose child class not satisfied with methods implemented in parent class, then child class can redefine same method in it but implementation can be different.
- We can override method and constructor.

overriding

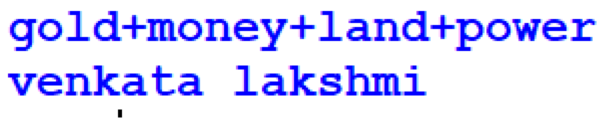
Example:

```
class parent:
        def properties(self):
                print(cash+money+land+gold+power)
        def marry(self):
                print(“venkata Lakshmi”)
class child(parent):
            def marry(self):
                super().marry()# it calls method in parent class
                print(“Alia batt”)
```

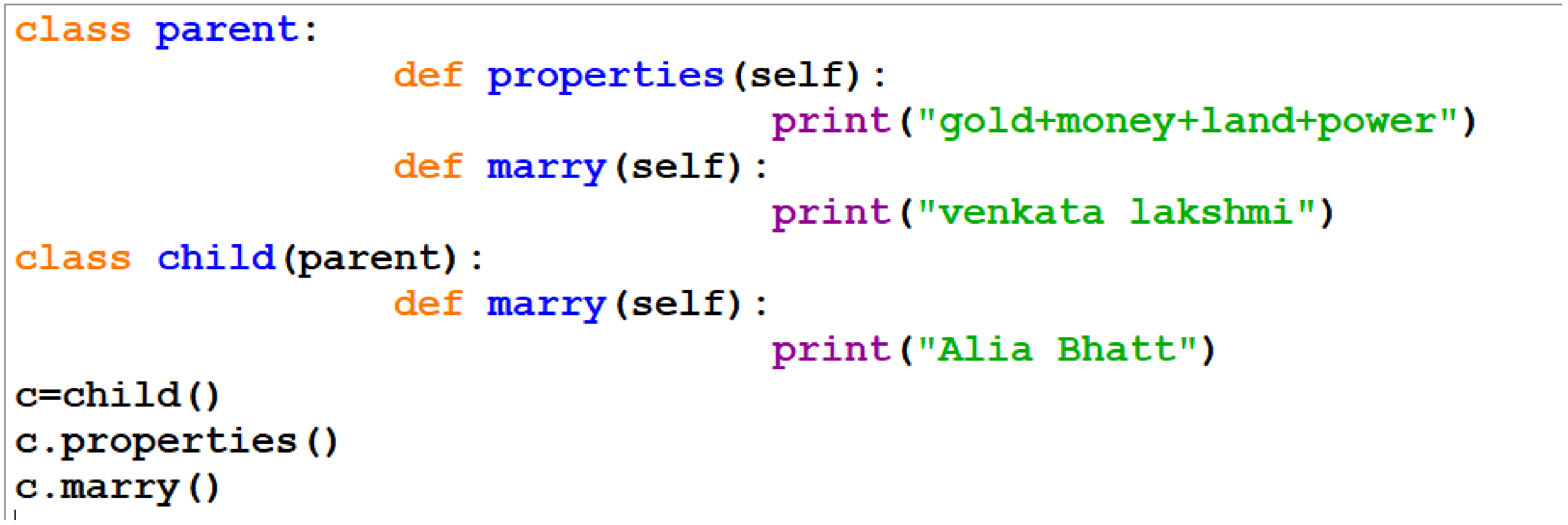


```python
class parent:
                def properties(self):
                                print("gold+money+land+power")
                def marry(self):
                                print("venkata lakshmi")
class child(parent):
                pass
c=child()
c.properties()
c.marry()
```

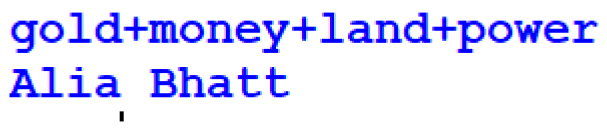
output



```
gold+money+land+power
venkata lakshmi
```



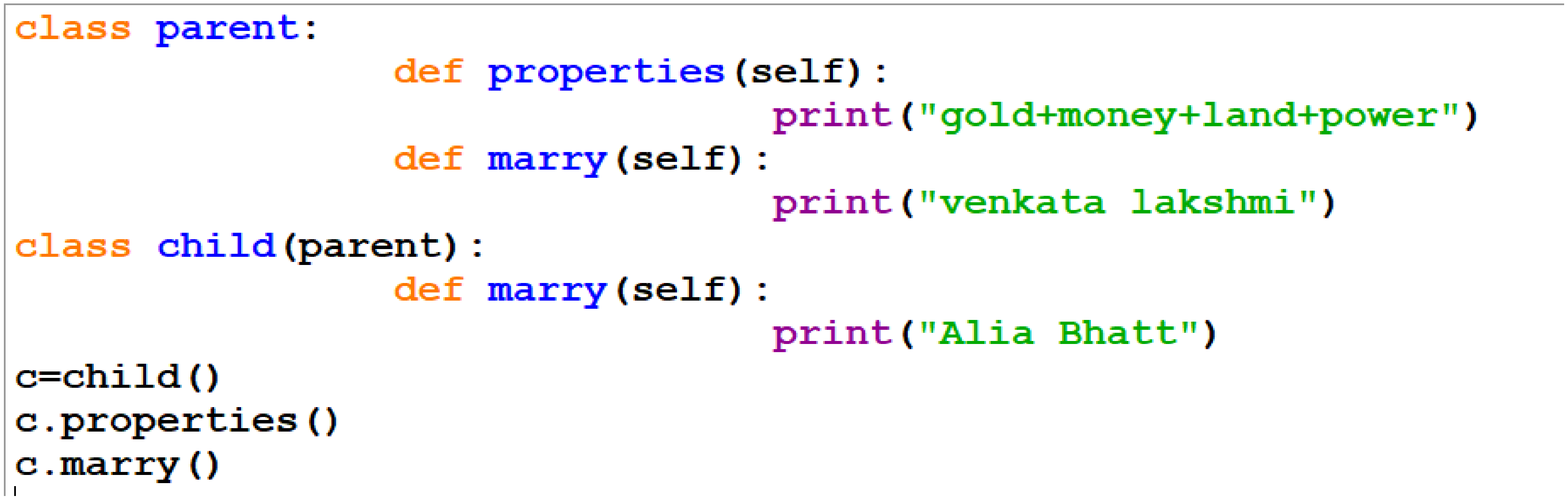
```
class parent:
        def properties(self):
                print("gold+money+land+power")
        def marry(self):
                print("venkata lakshmi")
class child(parent):
        def marry(self):
                print("Alia Bhatt")
c=child()
c.properties()
c.marry()
```



```
gold+money+land+power
Alia Bhatt
```

super() method

- From the child class if you want to call parent class methods use super() method.
- super() method should be define in overriding method.
- Below example child class wants to marry venkata lakshmi also.



```
class parent:
                def properties(self):
                                print("gold+money+land+power")
                def marry(self):
                                print("venkata lakshmi")
class child(parent):
                def marry(self):
                                print("Alia Bhatt")
c=child()
c.properties()
c.marry()
```

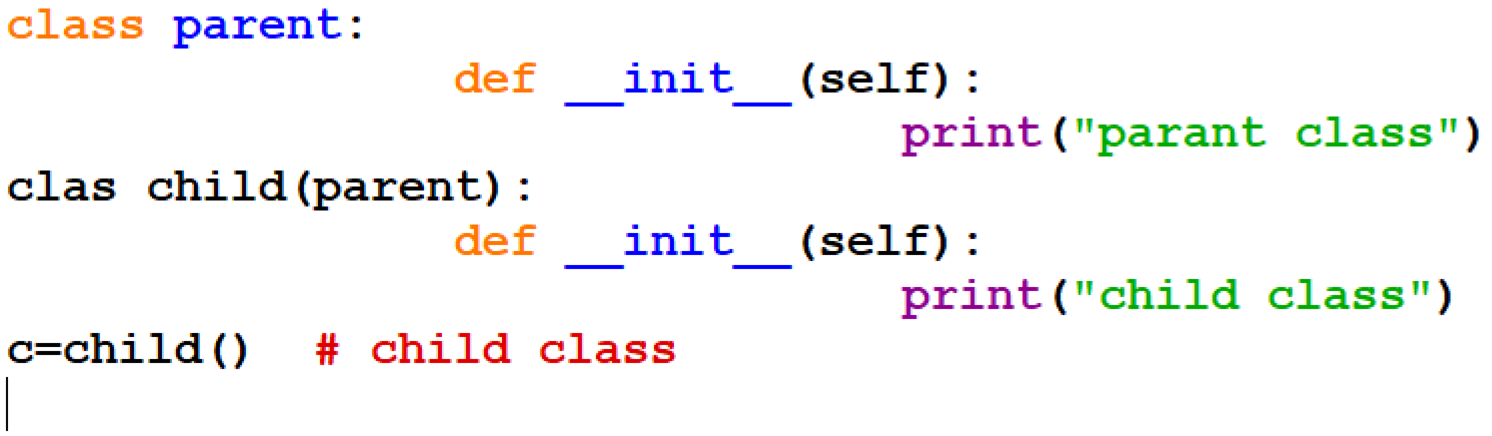
Example on super() method

Syntax : super().memberOfParentClass

```
class parent:
                def properties(self):
                                print("gold+money+land+power")
                def marry(self):
                                print("venkata lakshmi")
class child(parent):
                def marry(self):
                                super().marry()
                                print("Alia Bhatt")
c=child()
c.properties()
c.marry()
```

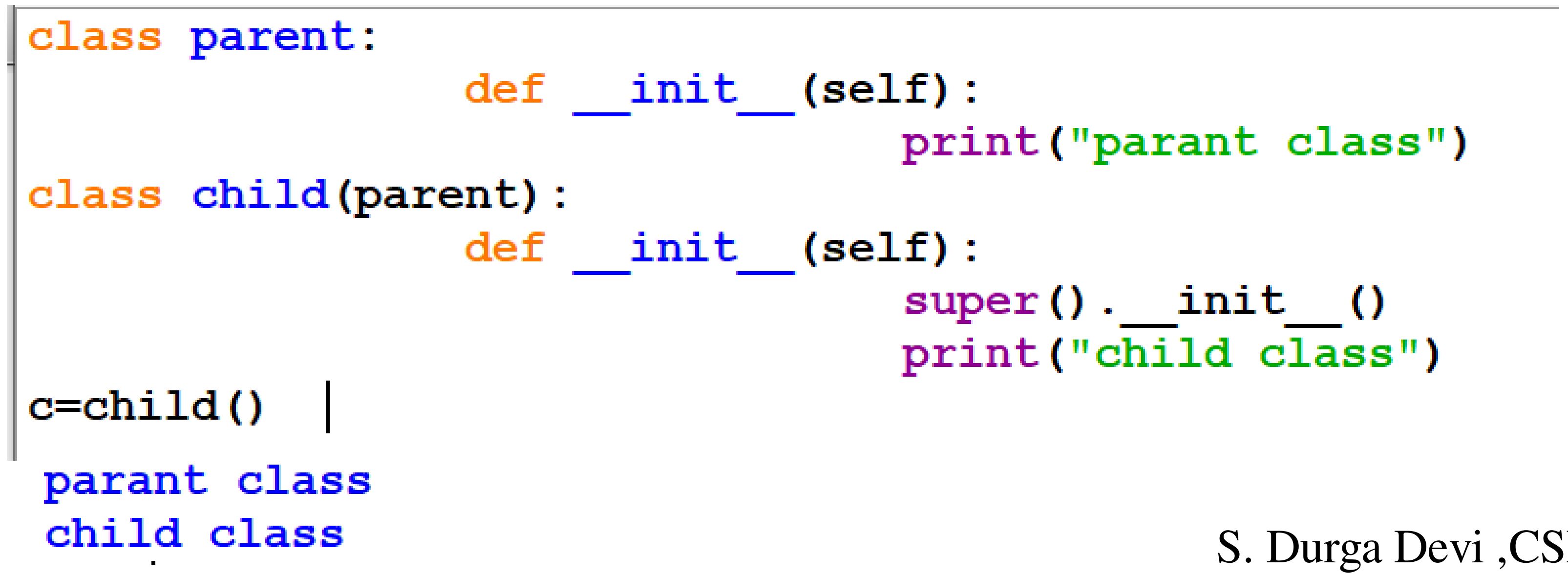
```
gold+money+land+power
venkata lakshmi
Alia Bhatt
```

Overriding constructor



```
class parent:
                def __init__(self):
                                print("parant class")
clas child(parent):
                def __init__(self):
                                print("child class")
c=child()  # child class
```

Call the parent class constructor from the child class constructor using super() method



```
class parent:
                def __init__(self):
                                print("parant class")
class child(parent):
                def __init__(self):
                                super().__init__()
                                print("child class")

c=child()
```

```
parant class
child class
```

- **Can we call methods in parent class from child class without using super()?**

- Yes we can access methods of parent class from child class without using super() method with following syntax

Parentclassname.methodname(self)

Example

```
class parent:
                def __init__(self):
                                print("parant class")
class child(parent):
                def __init__(self):
                                parent.__init__(self)
                                print("child class")

c=child()
```

```
parant class
child class
```

exercise

1. Write a program to convert meters to kilometers and kilo meters to meters.(refer page no 474 in text book, Example 11.8)
2. Write a program to override in operator (refer page no 472 in text book)

- Composition or containership
- Abstract classes and inheritance

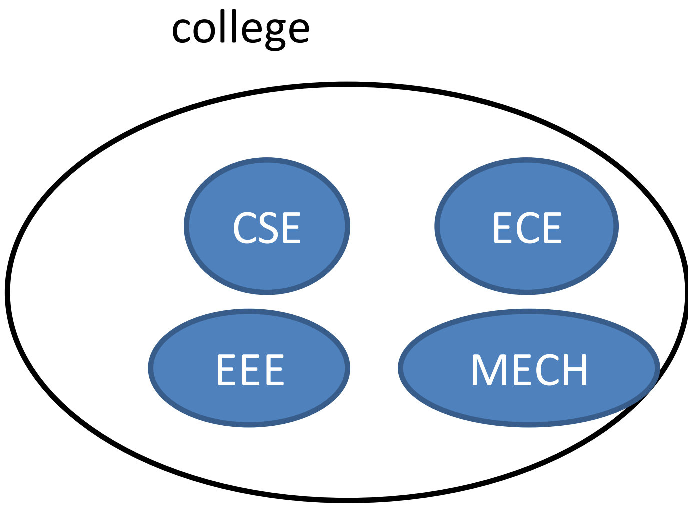
Composition or containership

- College vs Department (composition)

  within college several departments are there like CSE,ECE etc.

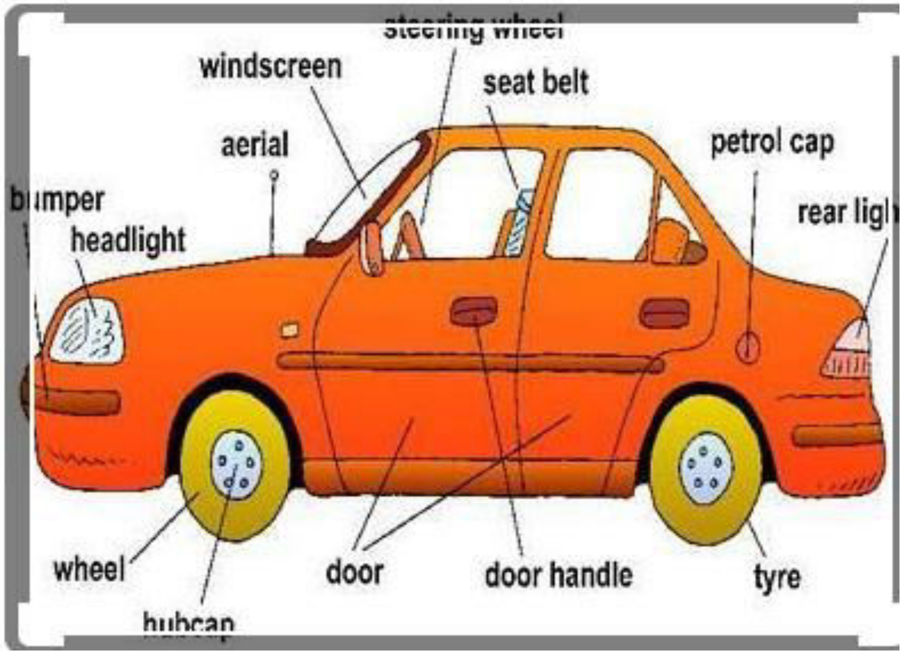
  - without college is there department?
  - department associated with the college.
  - here department strongly associated with the college.
  - college is also called container object
  - departments are contained objects.
  - college has departments(Has is relation).

- The process of creating complex objects from simple objects is called composition or containership.

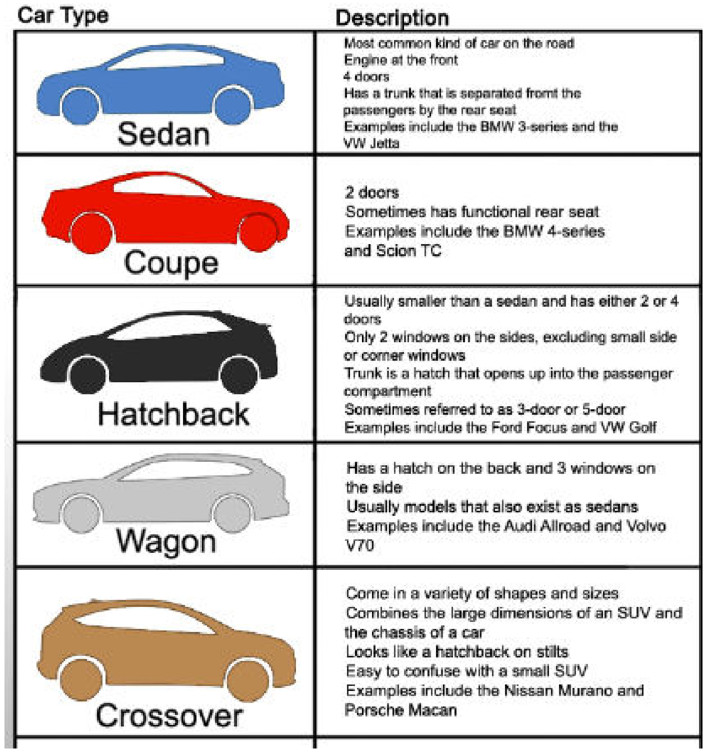


Here car is a composition which is composed of several parts like wheels, steering, engine, doors etc.

- If there is no car there are no parts of a car.



Car is an example of composition



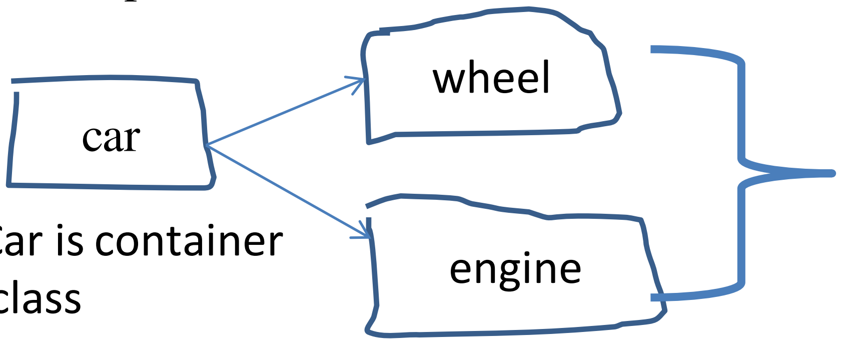
Car Type	Description
Sedan	Most common kind of car on the road Engine at the front 4 doors Has a trunk that is separated fromt the passengers by the rear seat Examples include the BMW 3-series and the VW Jetta
Coupe	2 doors Sometimes has functional rear seat Examples include the BMW 4-series and Scion TC
Hatchback	Usually smaller than a sedan and has either 2 or 4 doors Only 2 windows on the sides, excluding small side or corner windows Trunk is a hatch that opens up into the passenger compartment Sometimes referred to as 3-door or 5-door Examples include the Ford Focus and VW Golf
Wagon	Has a hatch on the back and 3 windows on the side Usually models that also exist as sedans Examples include the Audi Allroad and Volvo V70
Crossover	Come in a variety of shapes and sizes Combines the large dimensions of an SUV and the chassis of a car Looks like a hatchback on stilts Easy to confuse with a small SUV Examples include the Nissan Murano and Porsche Macan

Above fig is an example for inheritance

Composition

- Composition is elegant way of accessing parent class members from child class.
- Problem in inheritance is child class can have access all methods of parent class.
- By using composition we can explicitly choose what are the members of parent class that you need.
- Composition is defined in python by creating a class which as object of other class as a member.

example

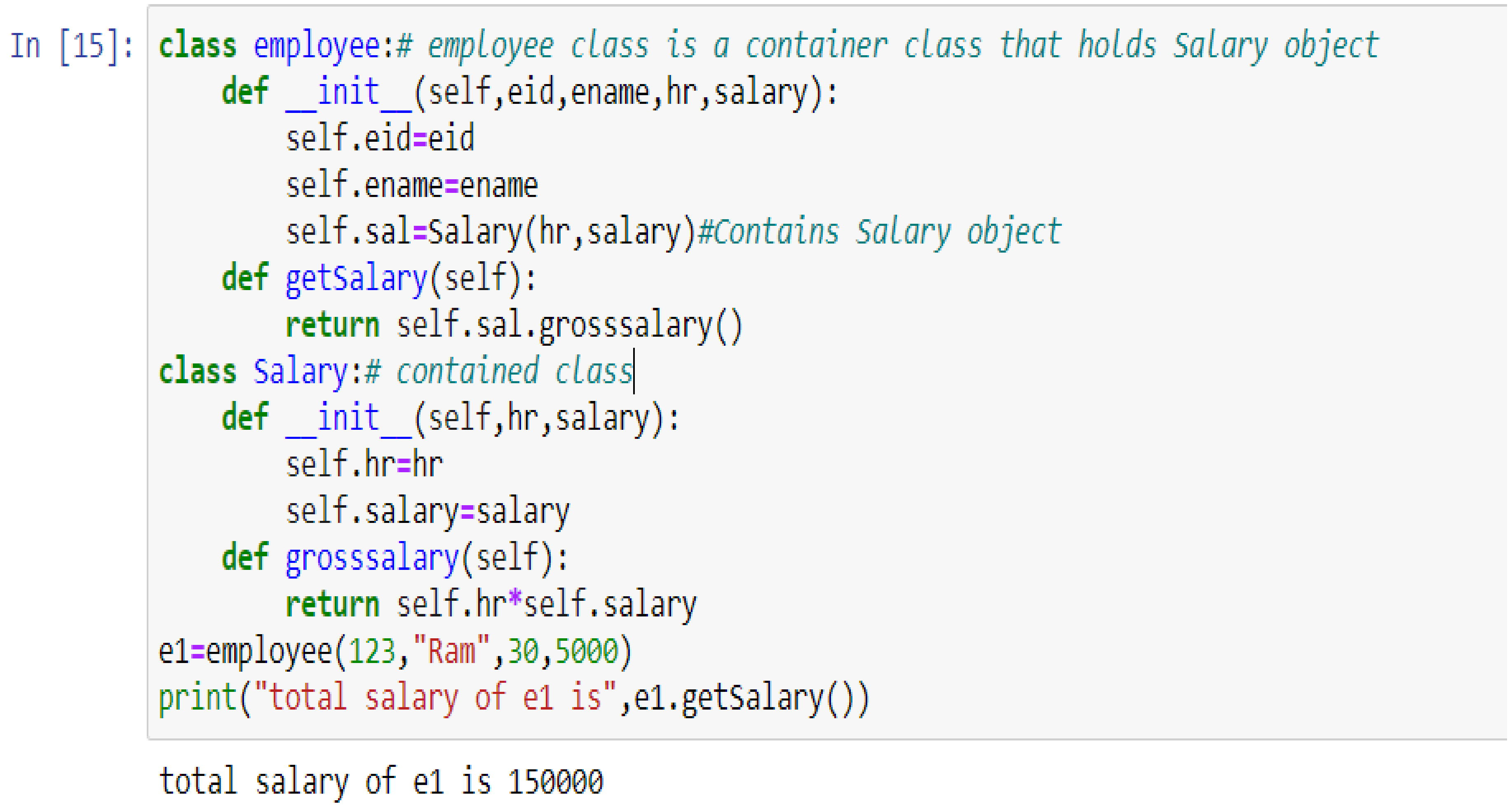


Car is container class

Contained classes

Container which contains objects

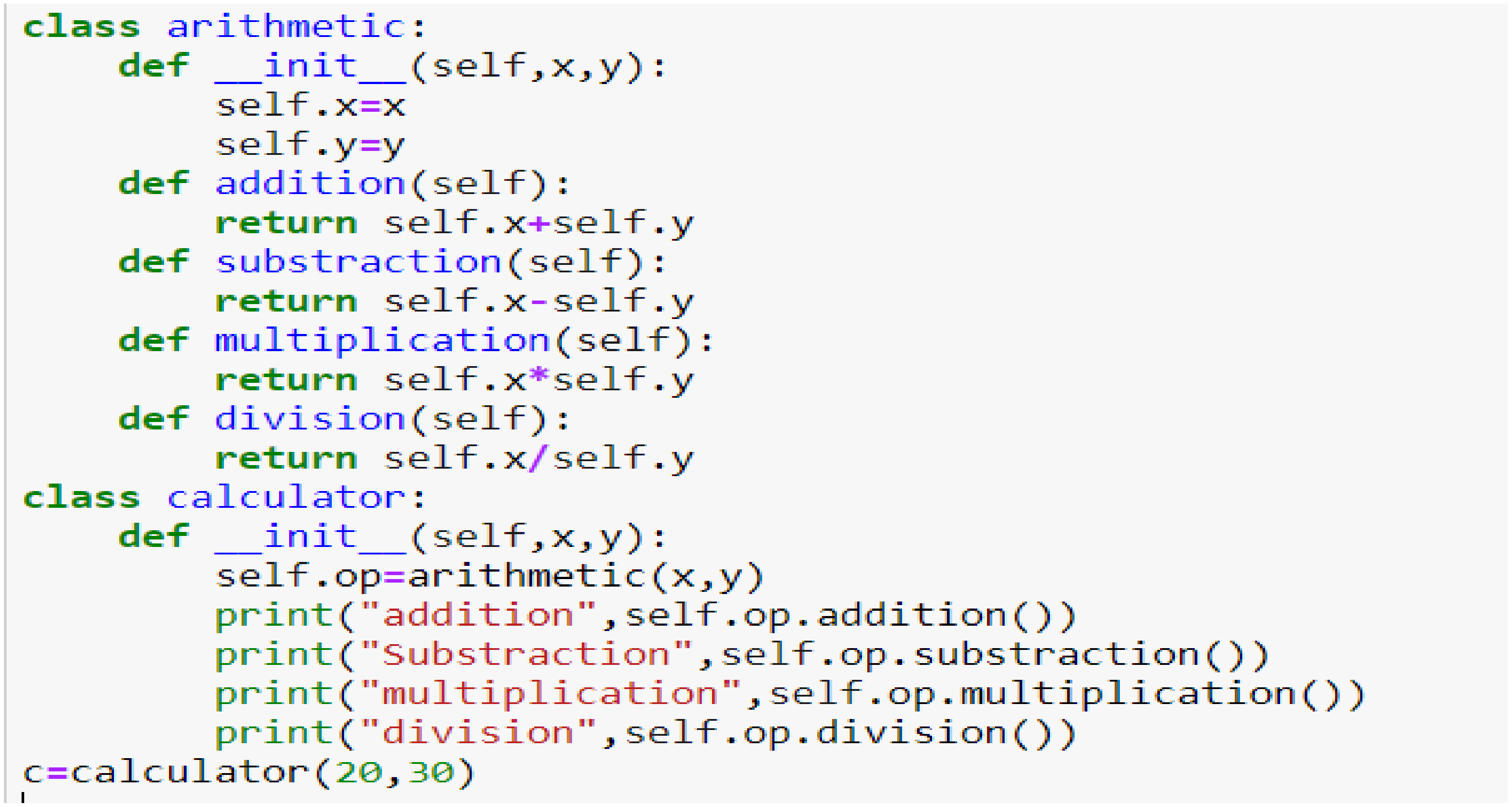
Composition example



```
In [15]: class employee:# employee class is a container class that holds Salary object
             def __init__(self,eid,ename,hr,salary):
                 self.eid=eid
                 self.ename=ename
                 self.sal=Salary(hr,salary)#Contains Salary object
             def getSalary(self):
                 return self.sal.grosssalary()
         class Salary:# contained class
             def __init__(self,hr,salary):
                 self.hr=hr
                 self.salary=salary
             def grosssalary(self):
                 return self.hr*self.salary
         e1=employee(123,"Ram",30,5000)
         print("total salary of e1 is",e1.getSalary())

         total salary of e1 is 150000
```

Write a program to implement calculator program using composition.



```python
class arithmetic:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def addition(self):
        return self.x+self.y
    def substraction(self):
        return self.x-self.y
    def multiplication(self):
        return self.x*self.y
    def division(self):
        return self.x/self.y
class calculator:
    def __init__(self,x,y):
        self.op=arithmetic(x,y)
        print("addition",self.op.addition())
        print("Substraction",self.op.substraction())
        print("multiplication",self.op.multiplication())
        print("division",self.op.division())
c=calculator(20,30)
```

Compare composition and inheritance

Inheritance	Composition or containership
1. Inheritance is used to model "IS A " Relation	**1. Composition used to model "HAS A" relation.**
2. A new class is created from existing class by extending it	**2. A class contains other class object as a member**
3. Derived class can have access all the members of its parent class	**3. Container class can have access its required members from the base class.**
4. Derived class can override methods of its base class	**4. Container class cannot override contained class methods.**

Abstract classes and inheritance

Abstract method: a method is only declared in a class but implementation not provided is called abstract method.

Example

```
class Parent:
        def  properties(self);
                print(“gold+money+land”)
        def marry(self):
                pass
```

Abstract class: if a class contains abstract method is called abstract class.

- A child class provides implementation of the abstract methods which are declared in parent class.

How abstract classes defined in python?

- By default python does not support the abstract classes.
- Python has several built in **Abstract Base Classes(ABC)** which are named as **'abc'** module.
- To define methods in base class as abstract method, abc module provides a decorator called as @abstractmethod.
- A class that extends abstract class provides implementation of abstract methods by overriding.
- **From abc module import 'ABC' class and '@abstractmethod' decorator in your python program**
- **Extend base class from 'ABC' class which is defined in 'abc' module.**
- **Define abstract methods by using decorator @abstractmethod.**

Example on abstract classes and methods

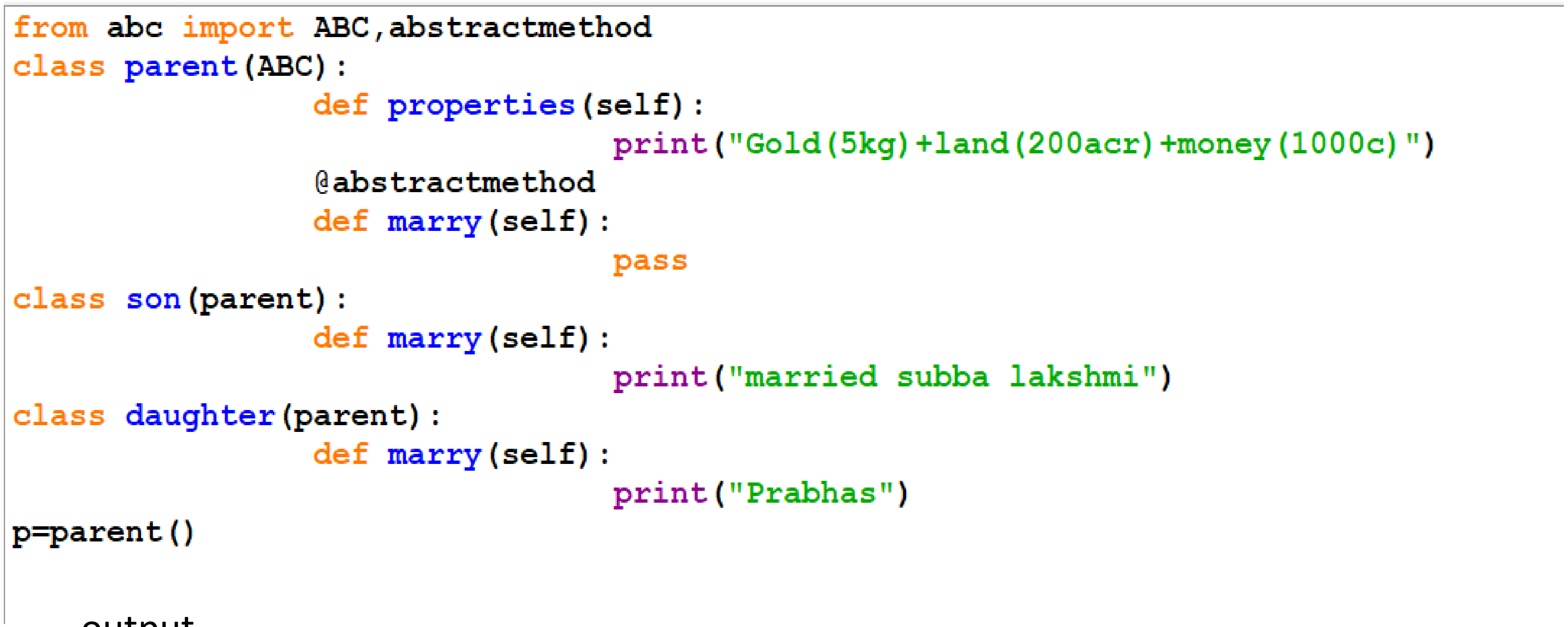
```python
from abc import ABC,abstractmethod
class parent(ABC):
                def properties(self):
                                print("Gold(5kg)+land(200acr)+money(1000c)")
                @abstractmethod
                def marry(self):
                                pass
class son(parent):
                def marry(self):
                                print("married subba lakshmi")
class daughter(parent):
                def marry(self):
                                print("Prabhas")
s=son()
s.marry()
d=daughter()
d.marry()
```

Example on abstract classes and methods

```
from abc import ABC,abstractmethod
class vehicle(ABC):
                @abstractmethod
                def wheels(self):
                                pass
                @abstractmethod
                def fueltype(self):
                                pass
class car(vehicle):
                def wheels(self):
                                print("car has 4 wheels")
                def fueltype(self):
                                print("Deisel")
class bike(vehicle):
                def wheels(self):
                                print("bike has 2 wheels")
                def fueltype(self):
                                print("Petrol")
c=car()
c.wheels()
c.fueltype()
b=bike()
b.wheels()
b.fueltype()
```

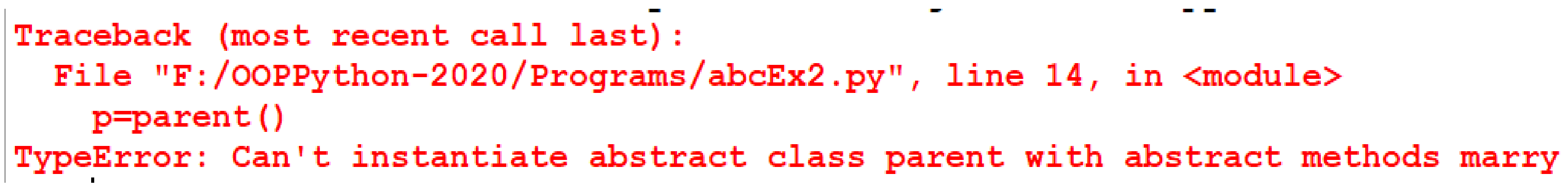
Can you instantiate abstract class?

Abstract classes cannot be instantiated.



```
from abc import ABC,abstractmethod
class parent(ABC):
                def properties(self):
                                print("Gold(5kg)+land(200acr)+money(1000c)")
                @abstractmethod
                def marry(self):
                                pass
class son(parent):
                def marry(self):
                                print("married subba lakshmi")
class daughter(parent):
                def marry(self):
                                print("Prabhas")
p=parent()
```

output



```
Traceback (most recent call last):
  File "F:/OOPPython-2020/Programs/abcEx2.py", line 14, in <module>
    p=parent()
TypeError: Can't instantiate abstract class parent with abstract methods marry
```

Summary on abstract classes

- Abstract method does not have implementation.
- Abstract class : The class has at least one abstract method.
- Abstract classes cannot be instantiated means objects cannot be created for the abstract classes.
- Abstract methods implementation provided by its child class.
- Abstract classes implemented in python by importing ‘ABC’ class and decorator ‘@abstractmethod’ form ‘abc’ module.
- Abstract class must extends the ‘ABC’ class.
- Abstract method defined with decorator ‘@abstractmethod’

Case study:

Calculate monthly salary of a particular employee by defining following classes.

1. Define a class ‘Employee’ which defines following methods.
    1. constructor with arguments name,salary
    2. magic method __mul__(self,x)

       __mul__() method overloads * operator to calculate employee salary based on the no of days he presented in particular month.
2. Define a class ‘employeeAttendance’ which defines constructor with argument is ‘days’

References

https://www.geeksforgeeks.org/abstract-classes-in-python/

https://www.tutorialspoint.com/abstract-base-classes-in-python-abc

https://www.journaldev.com/1325/composition-in-java-example