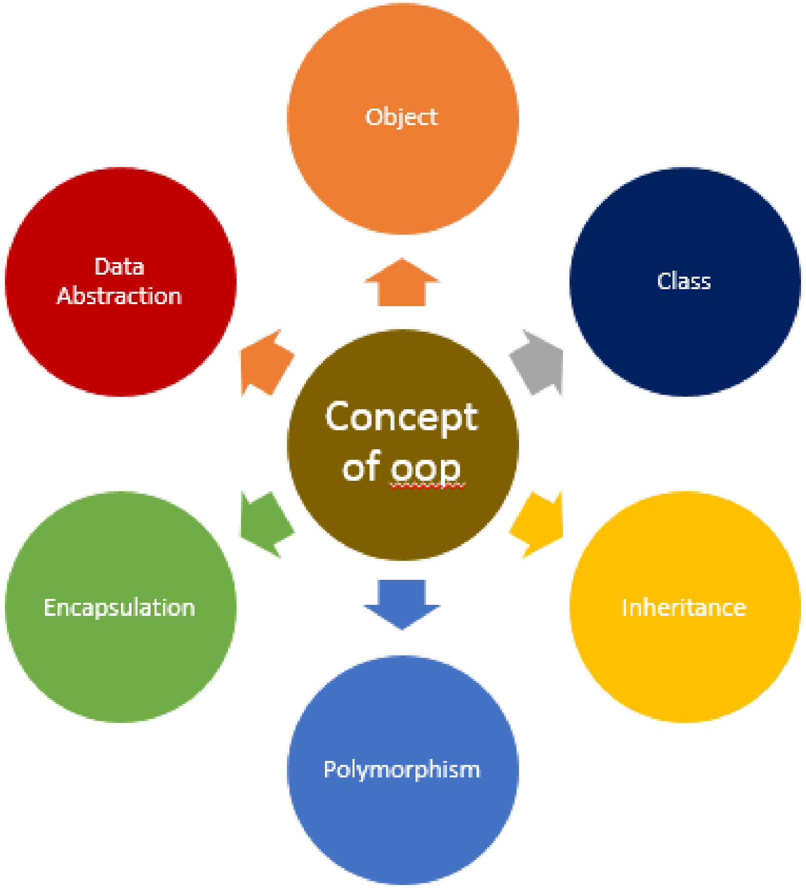
Classes and OOP



Why Class?

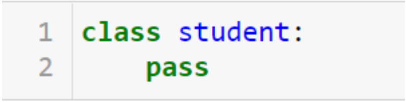
- classes are just a way to define new sorts of stuff, reflecting real objects in a program’s domain.
- If we implement class, we can model more of its real-world structure and relationships.
- Two aspects of OOPs very useful
- Inheritance
- Composition
- Operator overloading

Why Class?

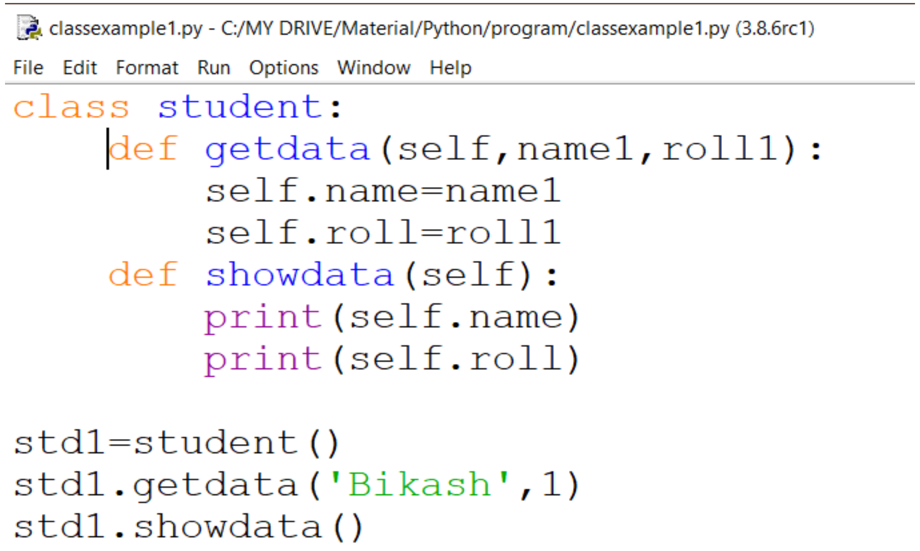
- classes are Python program units, just like functions and modules: they are another compartment for packaging logic and data.
- Principles of object-oriented programming system are given below.
- Class
- Object
- Method
- Inheritance
- Polymorphism
- Data Abstraction

How to Define a Class

- Class definitions start with the class keyword, followed by the name of the class and a colon.
- Any code that is indented below the class definition is considered part of the class's body.



```python
class student:
    pass
```

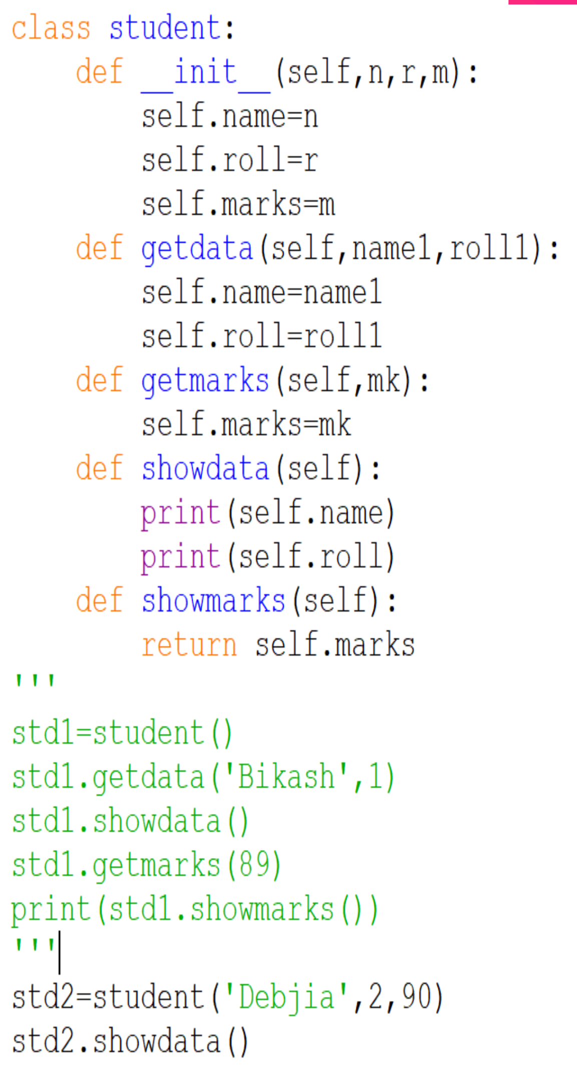


classexample1.py - C:/MY DRIVE/Material/Python/program/classexample1.py (3.8.6rc1)
File Edit Format Run Options Window Help

```python
class student:
        def getdata(self,name1,roll1):
                self.name=name1
                self.roll=roll1
        def showdata(self):
                print(self.name)
                print(self.roll)

std1=student()
std1.getdata('Bikash',1)
std1.showdata()
```

How to Define a Class

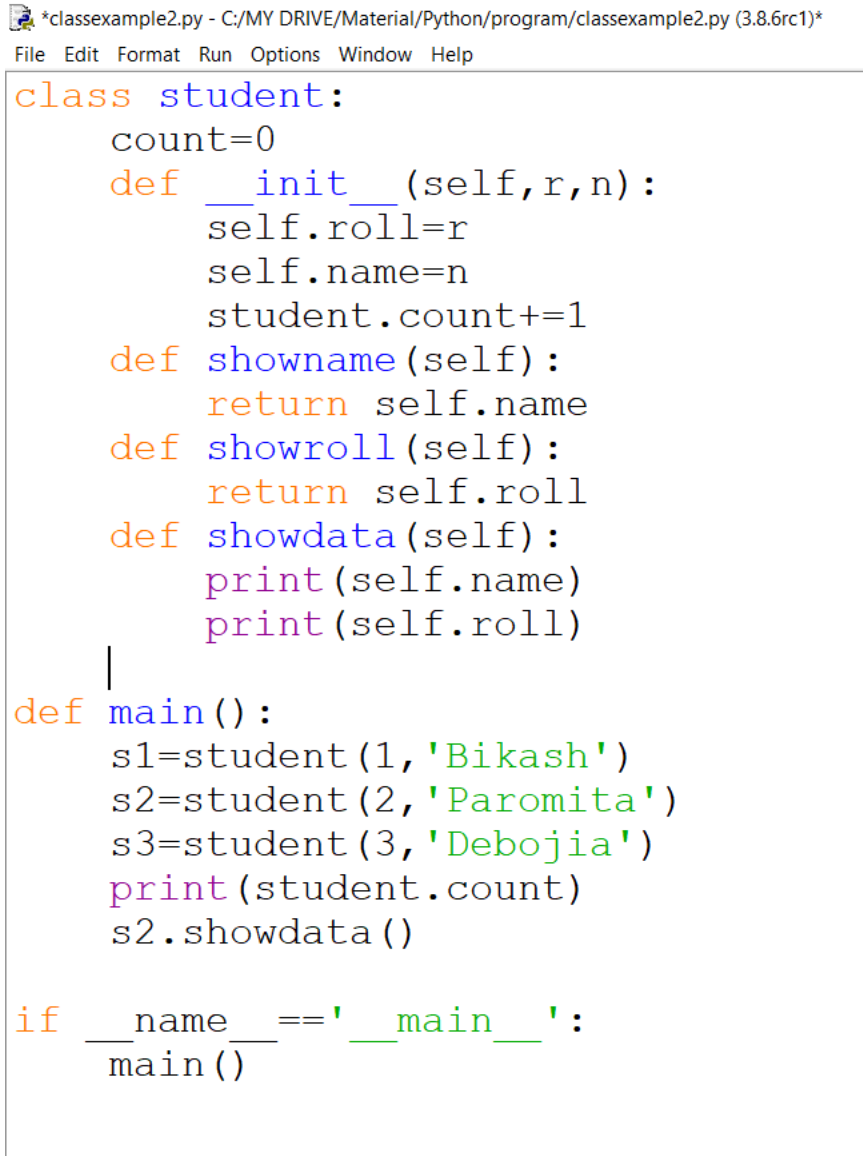
- The properties that all student objects must have can be defined in a method called __init__().
- Every time a new student object is created, __init__() sets the initial state of the object by assigning the values of the object’s properties.
- .__init__() initializes each new instance of the class.
- .__init__() any number of parameters, but the first parameter will always be a variable called self.
- When a new class instance is created, the instance is automatically passed to the self parameter in .__init__() so that new attributes can be defined on the object.



```
class student:
    def __init__(self,n,r,m):
        self.name=n
        self.roll=r
        self.marks=m
    def getdata(self,name1,roll1):
        self.name=name1
        self.roll=roll1
    def getmarks(self,mk):
        self.marks=mk
    def showdata(self):
        print(self.name)
        print(self.roll)
    def showmarks(self):
        return self.marks
'''
std1=student()
std1.getdata('Bikash',1)
std1.showdata()
std1.getmarks(89)
print(std1.showmarks())
'''
std2=student('Debjia',2,90)
std2.showdata()
```

Class Variable

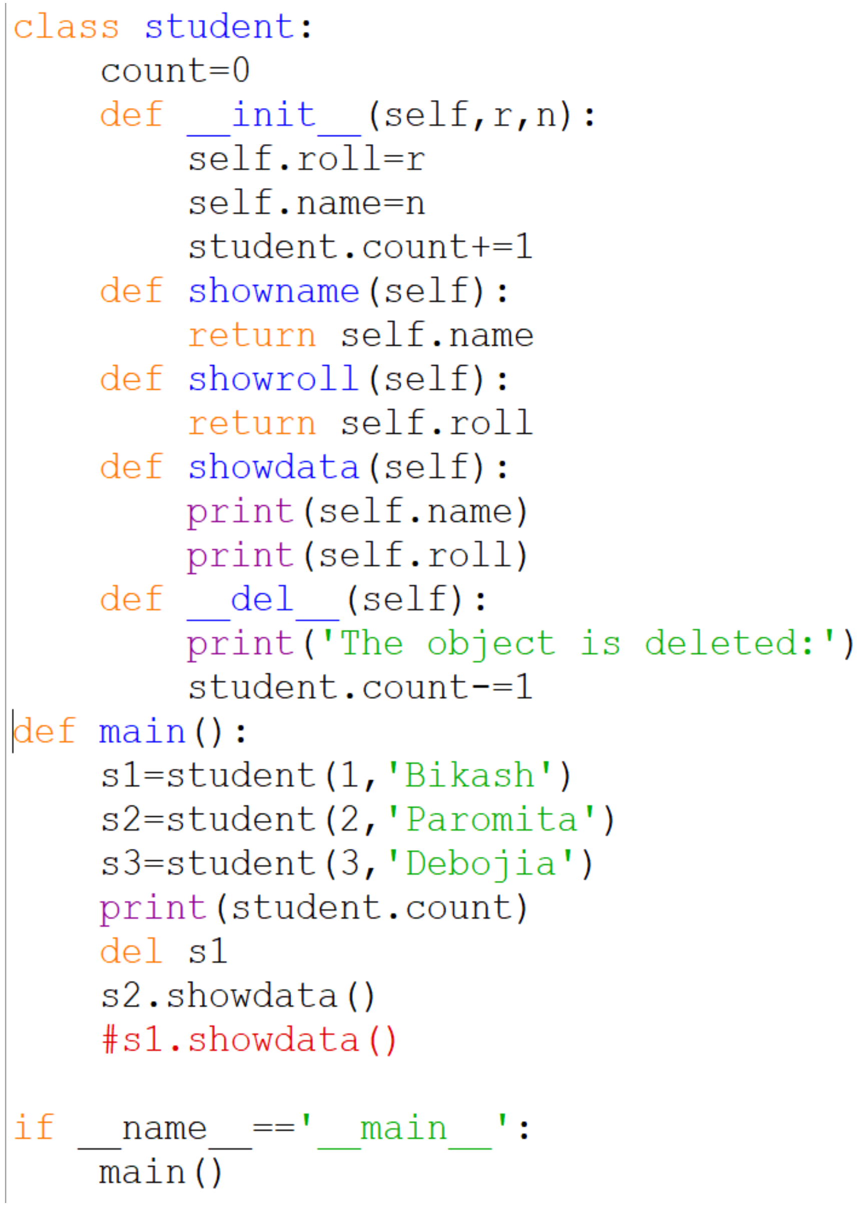
- Called as static variable.
- Class variable is initialized to zero.
- It is used to keep count of the number of student object created.


classexample2.py - C:/MY DRIVE/Material/Python/program/classexample2.py (3.8.6rc1)
File Edit Format Run Options Window Help

```python
class student:
    count=0
    def __init__(self,r,n):
        self.roll=r
        self.name=n
        student.count+=1
    def showname(self):
        return self.name
    def showroll(self):
        return self.roll
    def showdata(self):
        print(self.name)
        print(self.roll)

def main():
    s1=student(1,'Bikash')
    s2=student(2,'Paromita')
    s3=student(3,'Debojia')
    print(student.count)
    s2.showdata()

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

Destructor

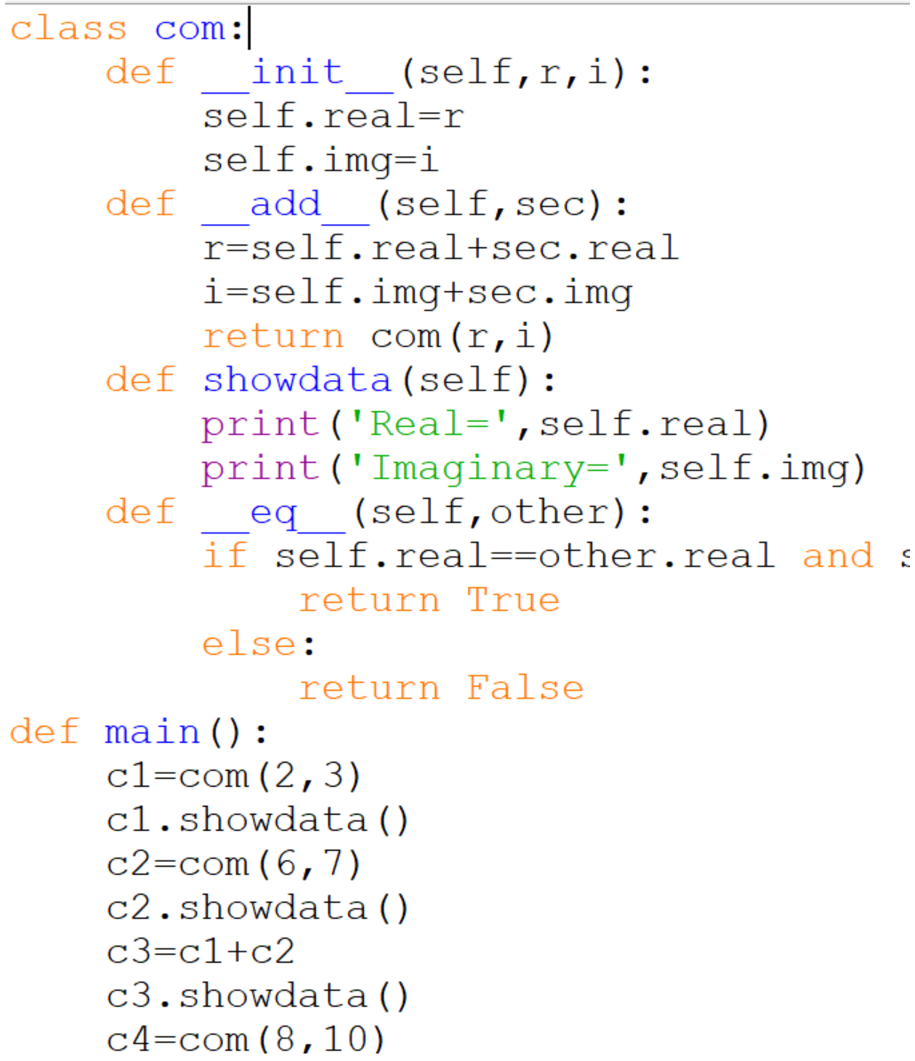
- When one object no more required, can delete the object.
- Destructor is used to deallocate the memory space for the object which is not required any more.
- __del__ method is use for destructor.
- Execution of *del* statement destroy the object from program namespace.



```
class student:
    count=0
    def __init__(self,r,n):
        self.roll=r
        self.name=n
        student.count+=1
    def showname(self):
        return self.name
    def showroll(self):
        return self.roll
    def showdata(self):
        print(self.name)
        print(self.roll)
    def __del__(self):
        print('The object is deleted:')
        student.count-=1
def main():
    s1=student(1,'Bikash')
    s2=student(2,'Paromita')
    s3=student(3,'Debojia')
    print(student.count)
    del s1
    s2.showdata()
    #s1.showdata()

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

Polymorphism

- A method /operator may be applied to objects of different types. This feature of object oriented programming is called polymorphism.
- When we add, subtract, multiply or divide two int or float objects using operators +,-,*,/, the corresponding Python special method __add, __sub__ etc gets invoked for the class(type) of objects.
- Python provides special methods such as __eq__, __lt__, __le__, __gt__, __ge__ for overloading comparison operators.



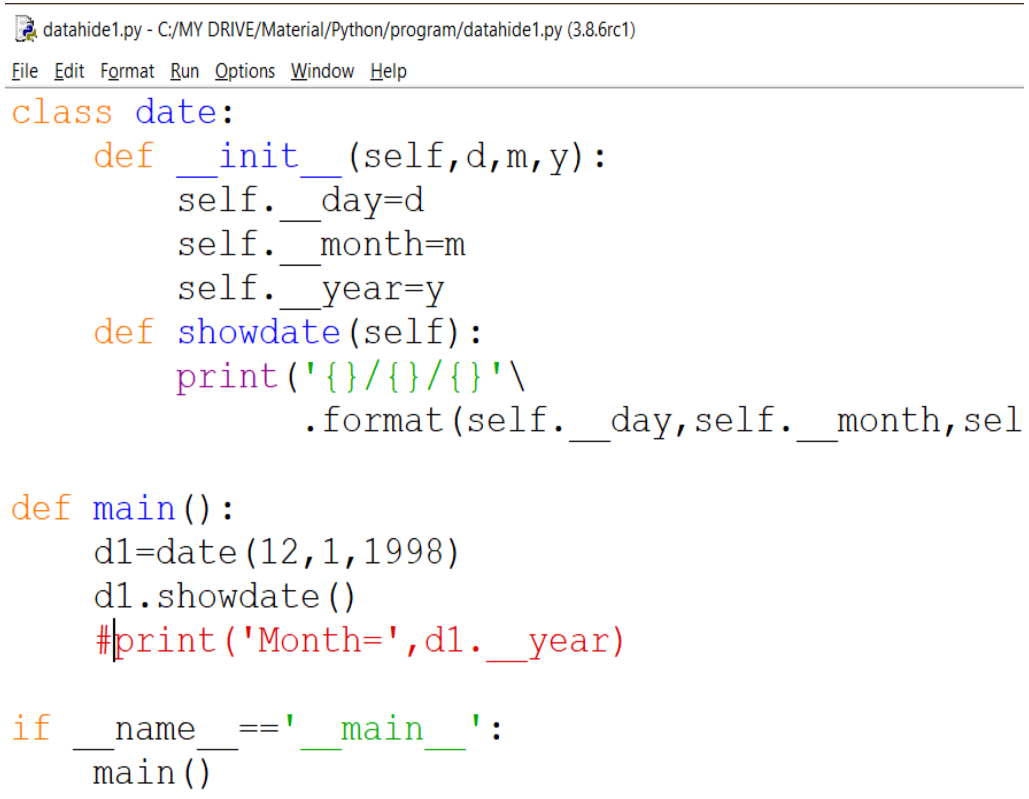
```
class com:
    def __init__(self,r,i):
        self.real=r
        self.img=i
    def __add__(self,sec):
        r=self.real+sec.real
        i=self.img+sec.img
        return com(r,i)
    def showdata(self):
        print('Real=',self.real)
        print('Imaginary=',self.img)
    def __eq__(self,other):
        if self.real==other.real and s
            return True
        else:
            return False
def main():
    c1=com(2,3)
    c1.showdata()
    c2=com(6,7)
    c2.showdata()
    c3=c1+c2
    c3.showdata()
    c4=com(8,10)
```

Encapsulation, Data hiding and Data Abstraction

- Encapsulation enables us to group together related data and its association functions under one name.
- Classes provide an abstraction where essential features of the real world.
- Accessing data and method outside of the class is a violation of principle of abstraction.
- Name mingling is a technique for defining private attributes.

Name Mingling

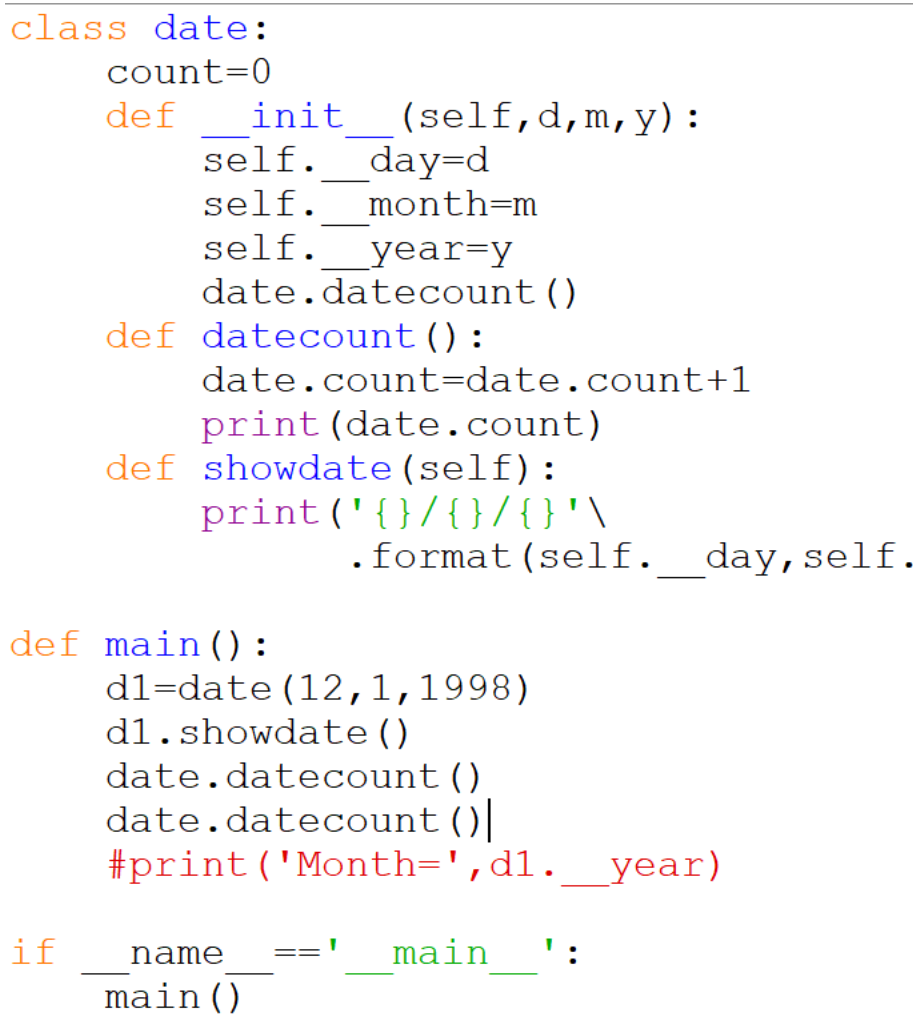
- One attribute can make private attribute by prefixing the attribute name by at least two consecutive underscore characters.
- The attribute name should not have more than one underscore character at the end
- This technique restrict the access of private members from outside the class, know as name mingling.


datahide1.py - C:/MY DRIVE/Material/Python/program/datahide1.py (3.8.6rc1)
File Edit Format Run Options Window Help

```
class date:
    def __init__(self,d,m,y):
        self.__day=d
        self.__month=m
        self.__year=y
    def showdate(self):
        print('{}/{}/{}'\
              .format(self.__day,self.__month,sel

def main():
    d1=date(12,1,1998)
    d1.showdate()
    #print('Month=',d1.__year)

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

Static method

- The method which passed the object implicitly(as *self*) is called instance method.
- The method which modify the class member does not require class object.
- A static method is invoked as an attribute of a class.



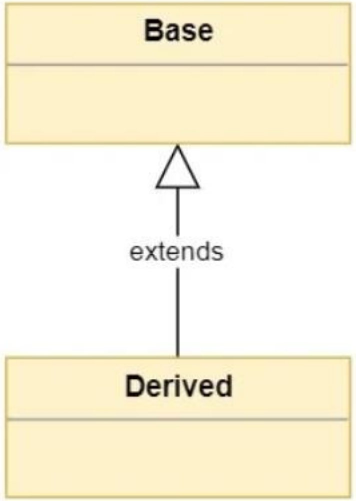
```
class date:
    count=0
    def __init__(self,d,m,y):
        self.__day=d
        self.__month=m
        self.__year=y
        date.datecount()
    def datecount():
        date.count=date.count+1
        print(date.count)
    def showdate(self):
        print('{}/{}/{}'\
              .format(self.__day,self.

def main():
    d1=date(12,1,1998)
    d1.showdate()
    date.datecount()
    date.datecount()
    #print('Month=',d1.__year)

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

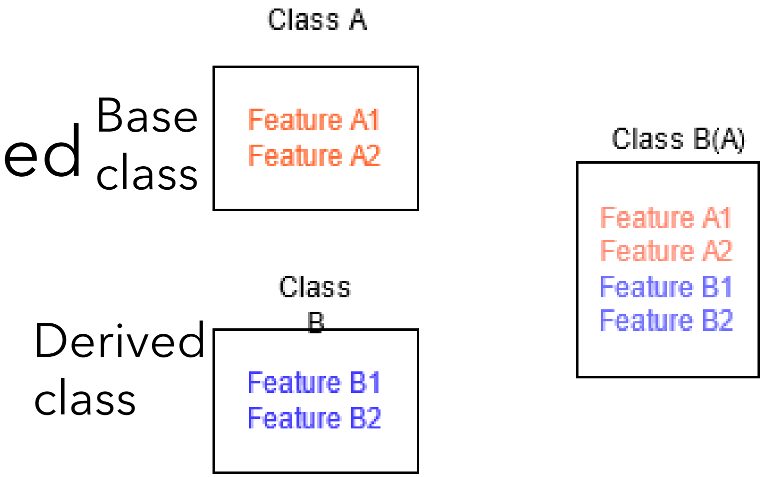
Inheritance and Composition

- **Inheritance** and **composition** are two major concepts in object oriented programming that model the relationship between two classes.
- They drive the design of an application and determine how the application should evolve as new features are added or requirements change.

What’s Inheritance?

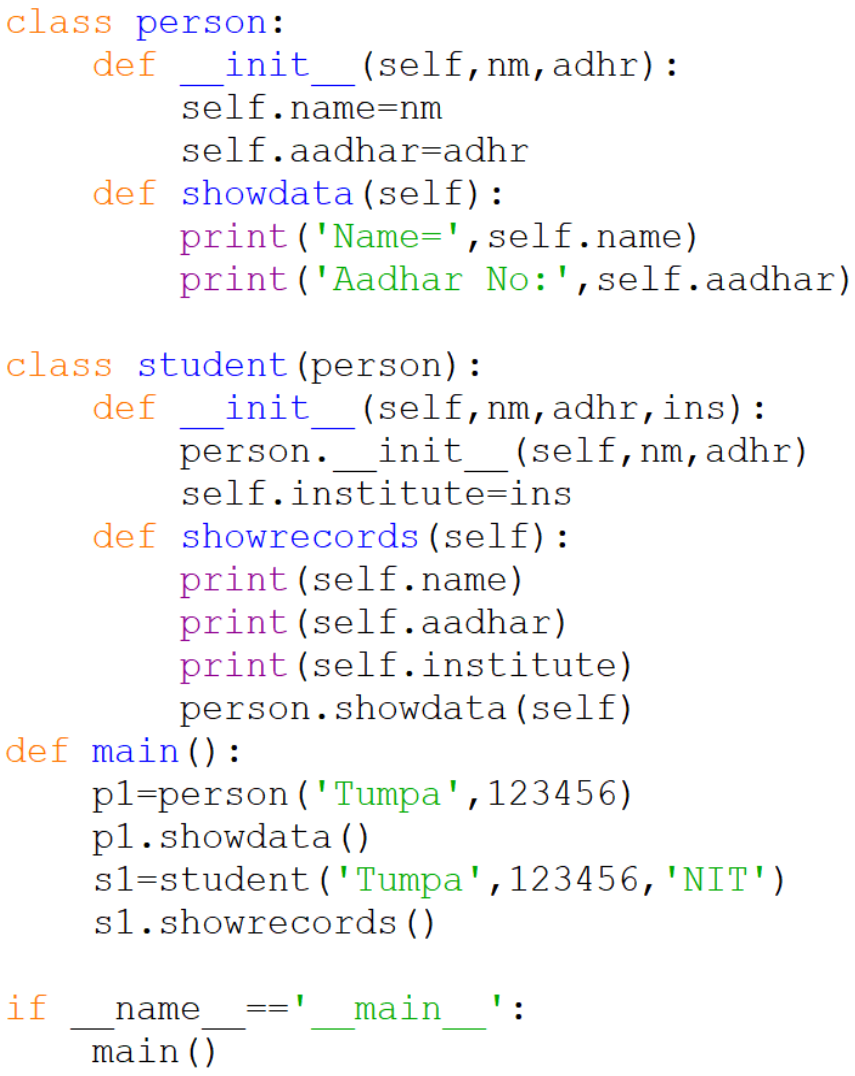
- **Inheritance** models, what is called an **is a** relationship.
- Derived class inherit all the properties of base class.
- Classes that inherit from another are called derived classes, subclasses, or subtypes.
- Classes from which other classes are derived are called base classes or super classes.





What's Inheritance?

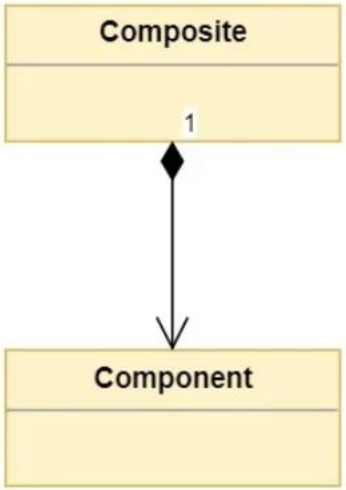
- The syntax of inheriting base class to derived class is
- $class\ base - class$:
- $\qquad pass$
- $class\ dervied - cls(base - cls)$:
- $\qquad Statements$

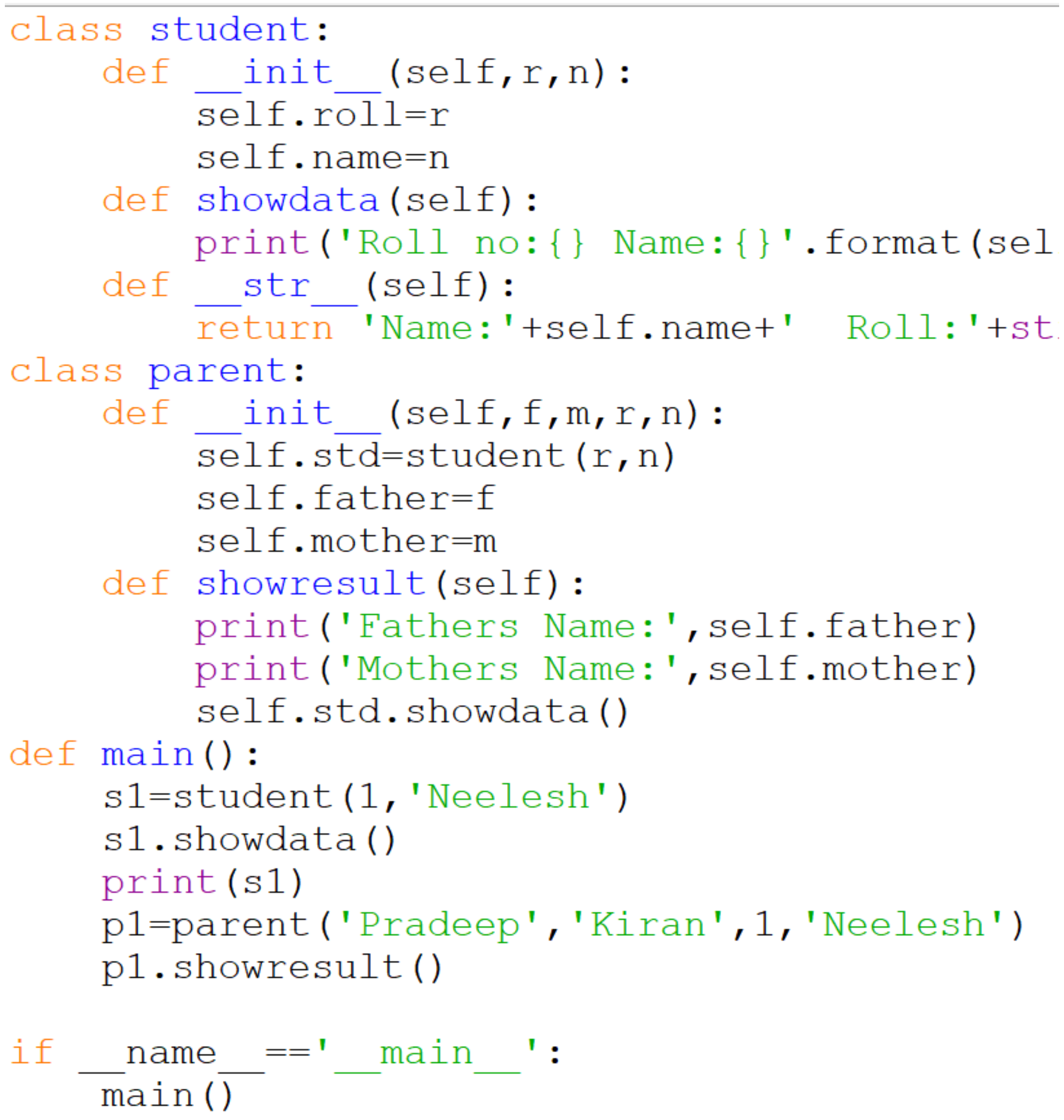


```
class person:
    def __init__(self,nm,adhr):
        self.name=nm
        self.aadhar=adhr
    def showdata(self):
        print('Name=',self.name)
        print('Aadhar No:',self.aadhar)

class student(person):
    def __init__(self,nm,adhr,ins):
        person.__init__(self,nm,adhr)
        self.institute=ins
    def showrecords(self):
        print(self.name)
        print(self.aadhar)
        print(self.institute)
        person.showdata(self)
def main():
    p1=person('Tumpa',123456)
    p1.showdata()
    s1=student('Tumpa',123456,'NIT')
    s1.showrecords()

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

What’s Composition?

- **Composition** is a concept that models a **has a** relationship.
- A class Composite can contain an object of another class Component.





```
class student:
    def __init__(self,r,n):
        self.roll=r
        self.name=n
    def showdata(self):
        print('Roll no:{} Name:{}'.format(sel
    def __str__(self):
        return 'Name:'+self.name+'  Roll:'+st
class parent:
    def __init__(self,f,m,r,n):
        self.std=student(r,n)
        self.father=f
        self.mother=m
    def showresult(self):
        print('Fathers Name:',self.father)
        print('Mothers Name:',self.mother)
        self.std.showdata()
def main():
    s1=student(1,'Neelesh')
    s1.showdata()
    print(s1)
    p1=parent('Pradeep','Kiran',1,'Neelesh')
    p1.showresult()

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

abc-Abstract Base Class

- An abstract method in a base class identifies the functionality that should be implemented by all its subclasses.
- Every subclass of the baseclass with override this method with its implementation.
- A class containing abstract method is called abstract class.

abc-Abstract Base Class

- This module provides the infrastructure for defining abstract base classes (ABCs) in Python
- an abstract base class can be created by simply deriving from ABC

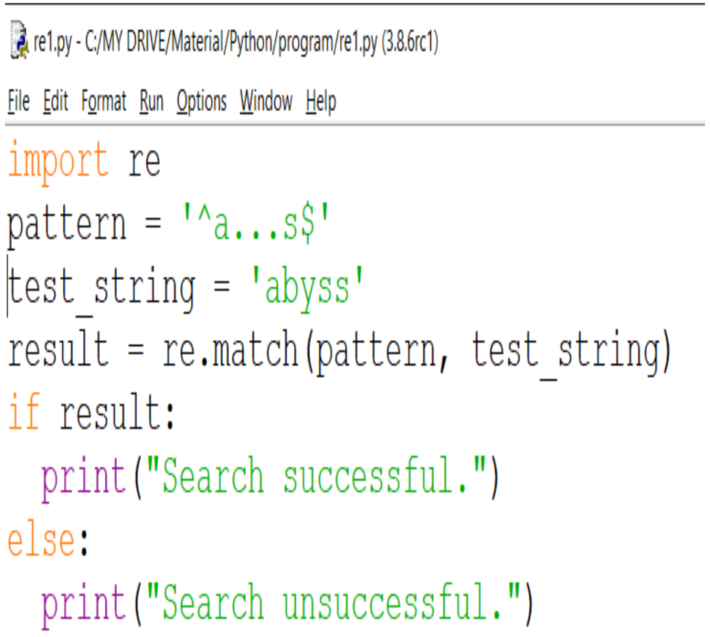
```
from abc import ABC
class MyABC(ABC):
    pass
```

- One may also define an abstract base class by passing the metaclass keyword and using ABCMeta directly

```
from abc import ABCMeta
class MyABC(metaclass=ABCMeta):
    pass
```

RegEx-regular Expression

- A Regular Expression (RegEx) is a sequence of characters that defines a search pattern.
- For example, ^a...s$
- The above code defines a RegEx pattern. The pattern is: any five letter string starting with a and ending with s.
- Python has a module named re to work with RegEx.



re1.py - C:/MY DRIVE/Material/Python/program/re1.py (3.8.6rc1)
File Edit Format Run Options Window Help

```
import re
pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)
if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```

MetaCharacters

- Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

- [] . ^ $ * + ? {} () \ |