**GANPAT UNIVERSITY**

**U. V. PATEL COLLEGE OF ENGINEERING**

**B.Tech CE/IT Semester IV**

**2CEIT404: Python Programming**

**Practical-8: Object oriented programming with python**

**[1] Create a class Employee with data members: name, department and salary. Use constructor to initialize values and display() method for printing information of three employees.**

**Program:**

*class* Employee:

*def* \_\_init\_\_(*self* , *name*, *department*, *salary*):

*self*.name = *name*

*self*.department = *department*

*self*.salary = *salary*

*def* dispalyDetails(*self*):

        print("name", *self*.name,"Department",*self*.department,"Salary",*self*.salary)

e1 = Employee("Vandan","CE",50000)

e2 = Employee("Jaydip","IT",45000)

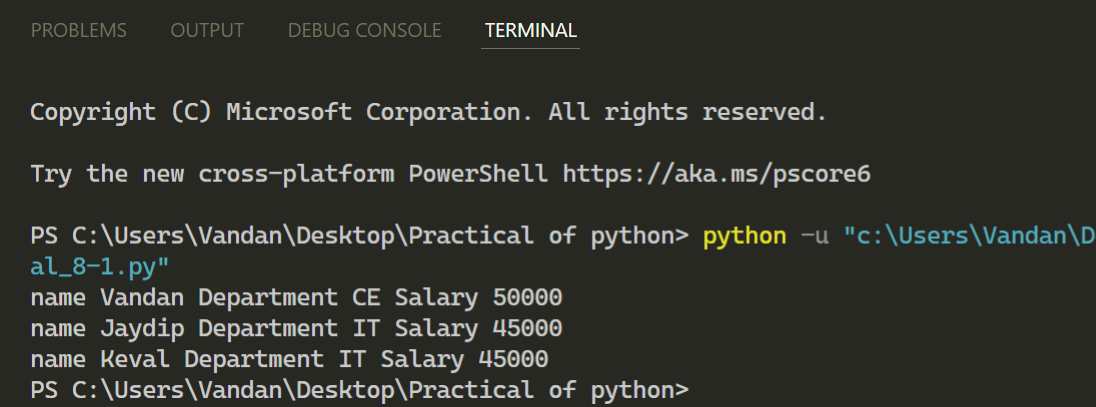
e3 = Employee("Keval","IT",45000)

e1.dispalyDetails()

e2.dispalyDetails()

e3.dispalyDetails()

**Output:**

****

**[2] Write a program to create class Student with following attributes: instance variables enrollment\_no, name and branch; instance methods get\_value() and print\_value(); class variable cnt; static method show(). Variable cnt counts number of instances created and show() method displays value of cnt.**

**Program :**

*class* Student:

    count =0

*def* \_\_init\_\_(*self*):

        Student.count += 1

*def* get\_values(*self*,*enrollment\_no*,*name*,*branch*):

*self*.enrollment\_no = *enrollment\_no*

*self*.name = *name*

*self*.branch = *branch*

*def* print\_value(*self*):

        print("Enrollment\_no",*self*.enrollment\_no,"Name",*self*.name,"Branch",*self*.branch)

    @staticmethod

*def* display1():

            print("Total instance created = ",Student.count)

S1 = Student()

S1.get\_values(130,"Vandan Patel","CE")

S1.print\_value()

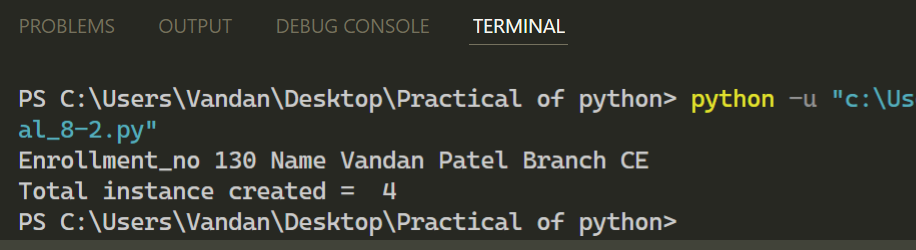
s2 = Student()

s3 = Student()

s4 = Student()

Student.display1()

**Output:**

****

**[3] Write a program to overload \*\* (exponential) operator.**

**Program :**

*class* exp:

*def* \_\_init\_\_(*self*,*a*):

*self*.a = *a*

*def* \_\_pow\_\_(*self*,*o*):

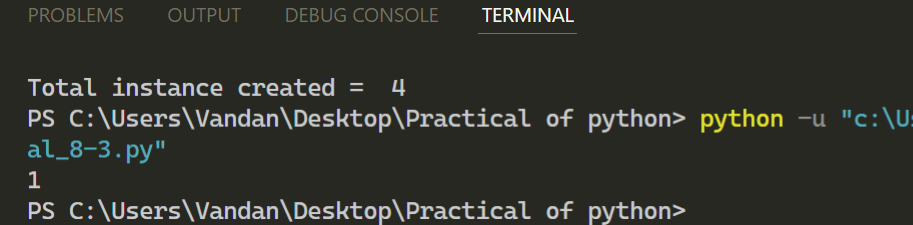
        return *self*.a \*\**o*.a

ob1 =exp(1)

ob2 = exp(2)

print(ob1 \*\* ob2)

**Output:**

****

**[4] Create class Hospital having attributes patient\_no, patient\_name and disease\_name and an instance p1. Show use of methods getattr(), setattr(), delattr(), and hasattr() for p1. Display values of attributes \_\_dict\_\_, \_\_doc\_\_, \_\_name\_\_, \_\_module\_\_, \_\_bases\_\_ with respect to class Hospital. Delete instance p1 in the end.**

**Program :**

*class* Hospital:

*def* \_\_init\_\_(*self*,*pat\_no*,*pat\_name*,*dis\_name*):

*self*.pat\_no=*pat\_no*

*self*.pat\_name=*pat\_name*

*self*.dis\_name=*dis\_name*

p1=Hospital(130,"Vandan Patel","Corana")

print(getattr(p1,'pat\_name'))

setattr(p1,'pat\_no',125)

print(hasattr(p1,'pat\_name'))

print(Hospital.\_\_dict\_\_)

print(Hospital.\_\_doc\_\_)

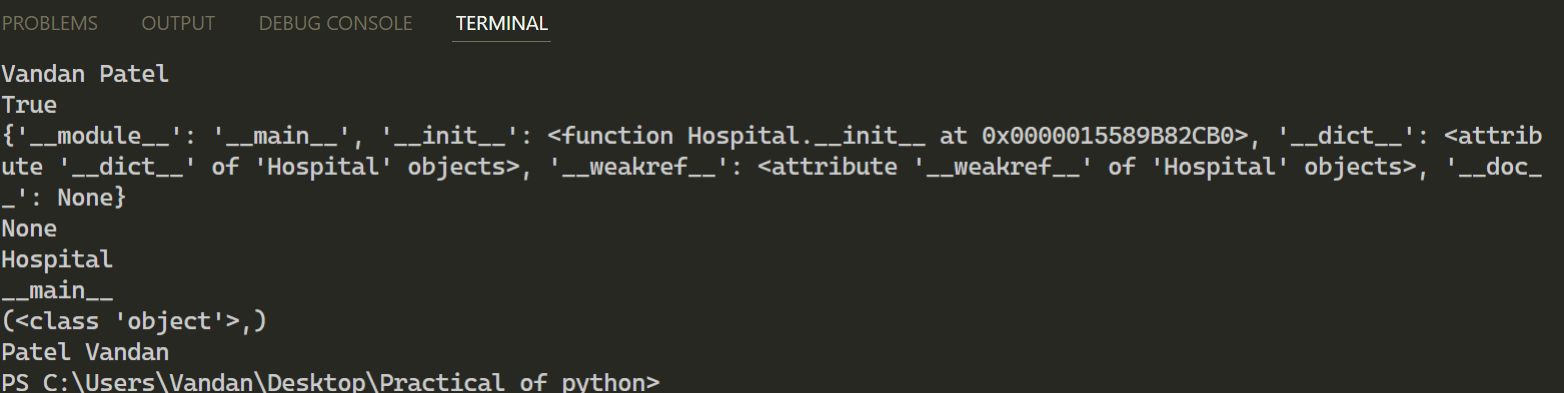
print(Hospital.\_\_name\_\_)

print(Hospital.\_\_module\_\_)

print(Hospital.\_\_bases\_\_)

print("Patel Vandan")

**Output:**

****

**[5] Design a class Lion having method roar() and a class Cub having method play() which inherits class Lion. Use instance of Cub called- simba to access methods roar() and play(). Define public attribute legs, protected attribute ears and private attribute mane of class Lion. Show accessibility of these variables according to their scope.**

**Program :**

*class* Lion:

*def* \_\_init\_\_(*self*,*legs*,*ears*,*name*):

*self*.legs=*legs*

*self*.\_ears=*ears*

*self*.\_\_name=*name*

*def* roar(*self*):

  print("Loud Roar")

*class* Cub(Lion):

*def* \_\_init\_\_(*self*, *legs*, *ears*, *name*):

  super().\_\_init\_\_(*legs*, *ears*, *name*)

*def* play(*self*):

  print("Love Playing")

c=Cub(3, 2, 'x')

c.play()

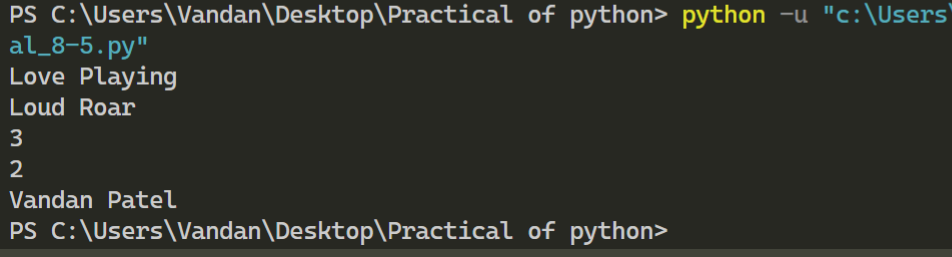
c.roar()

print(c.legs)

print(c.\_ears)

print("Vandan Patel")

**Output:**

****

**[6] Class Person with attributes- name and age is inherited by class SportPerson with attribute sport\_name. Use appropriate \_\_init\_\_() method for both classes. Call parent \_\_init\_\_() method from child \_\_init\_\_() method with the help of (A) super() method (B) parent class name.**

**Program :**

*class* Person:

*def* \_\_init\_\_(*self*,*name*,*age*):

*self*.name=*name*

*self*.age=*age*

*class* SpotPerson(Person):

*def* \_\_init\_\_(*self*,*name*,*age*,*sports\_name*):

  super().\_\_init\_\_(*name*,*age*)

*self*.sports\_name=*sports\_name*

*def* print(*self*):

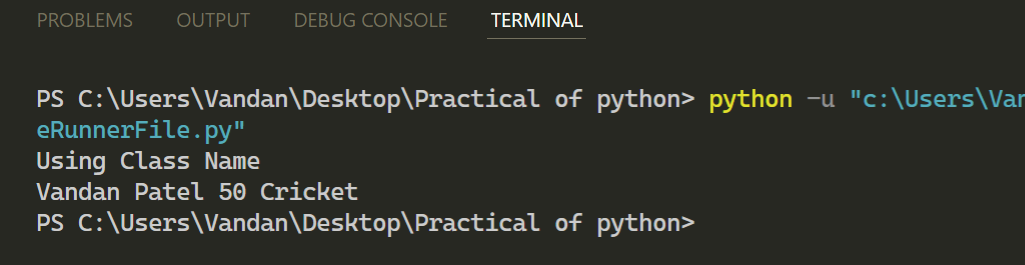
  print(*self*.name,*self*.age,*self*.sports\_name)

x=SpotPerson("Vandan Patel",50,"Cricket")

print("Using Class Name")

x.print()

**Output:**

****

**[7] Write programs to implement following scenarios where A, B, C, D, E and F are classes and check() is a method. In both scenarios, which check() method is called, when we execute statement- E().check()**

**Program :**

*class* a:

*def* check(*self*):

  print("Vandan1")

*class* b:

*def* check(*self*):

  print("Vandan2")

*class* c:

*def* check(*self*):

  print("Vandan3")

*class* d:

*def* check(*self*):

  print("Vandan4")

*class* e:

*def* check(*self*):

  print("Vandan5")

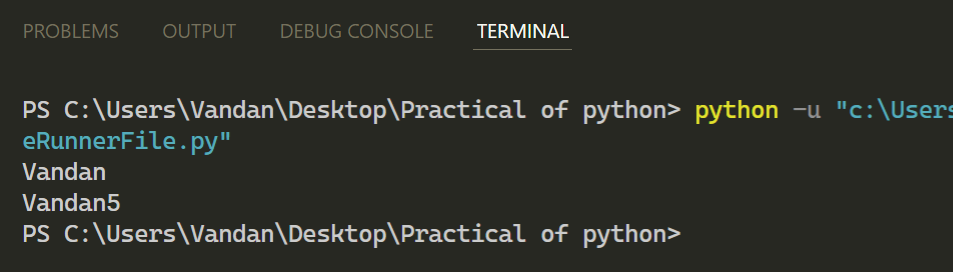
*def* check():

 print("Vandan")

c=check()

e().check()

**Output:**

****

**[8] Write a program in which Python and Snake sub classes implement abstract methods- crawl() and sting() of the super class Reptile. What is the output of following statements? Program :**

*class* Reptile:

*def* crawl():

  pass

*def* string():

  pass

*class* python(Reptile):

*def* crawl():

   pass

*def* string():

  pass

*class* snake(Reptile):

*def* crawl():

  pass

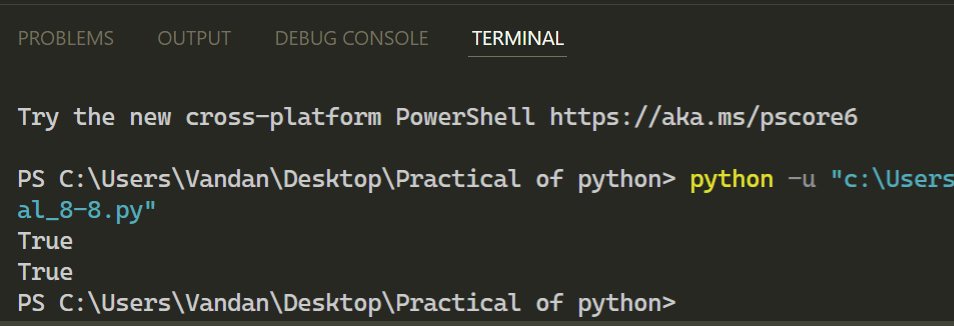
*def* string():

  pass

print(issubclass(python, Reptile))

print(isinstance(snake(), Reptile))

**Output:**

****