GANPAT UNIVERSITY
U. V. PATEL COLLEGE OF ENGINEERING
B.Tech CE/IT Semester IV
2CEIT404: Python Programming

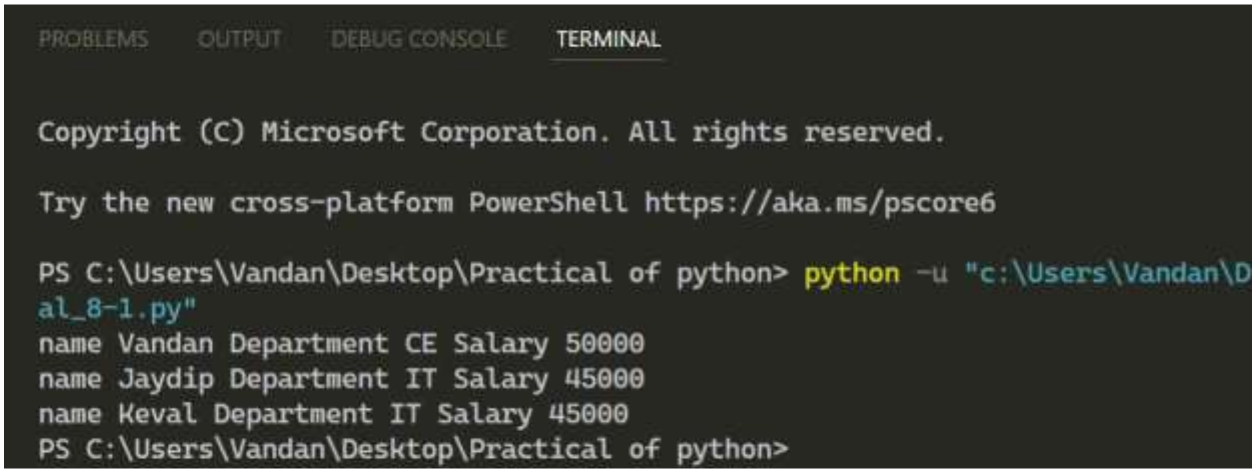
Practical-8: Object oriented programming with python

[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:

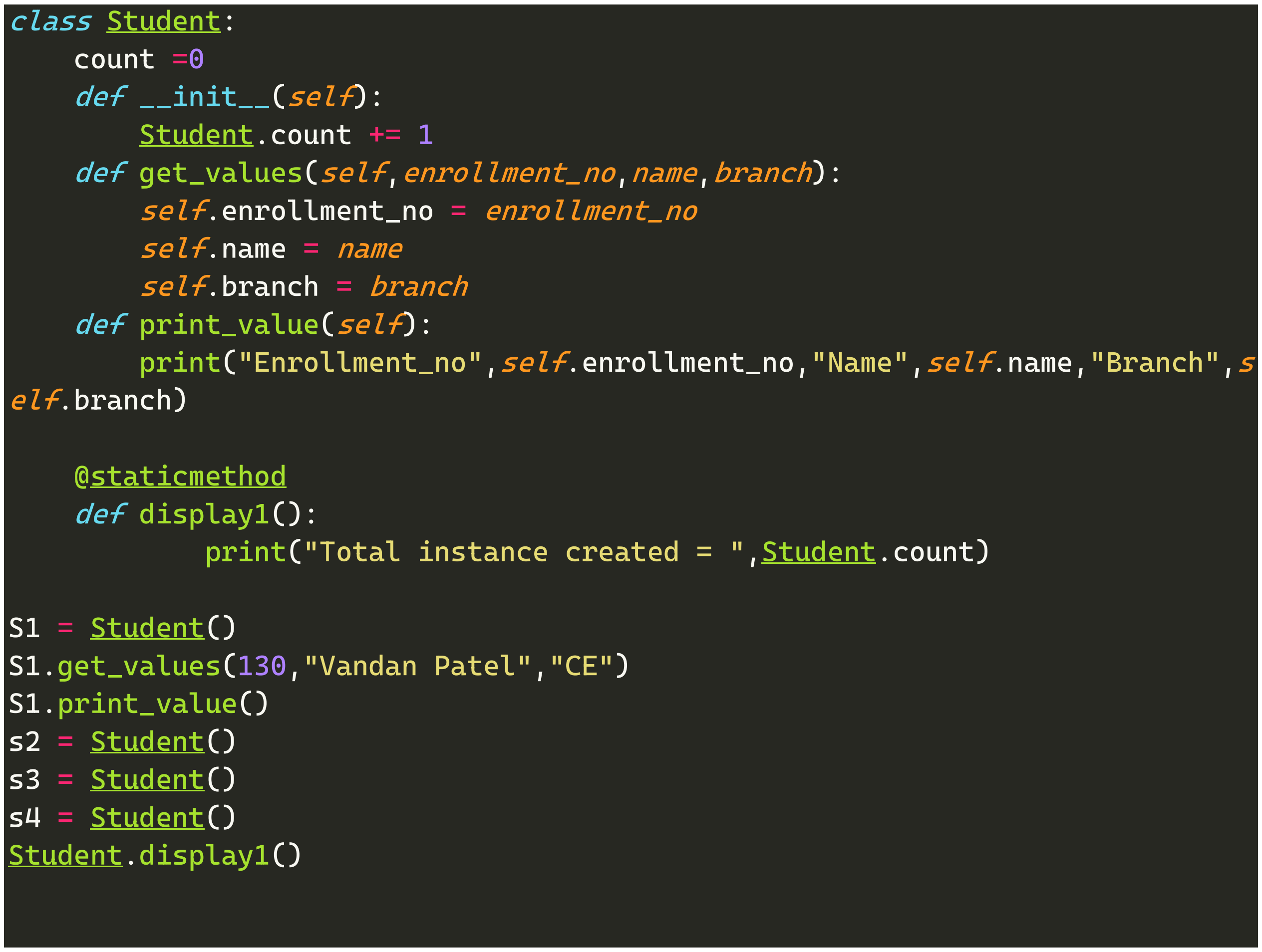


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\Users\Vandan\D
al_8-1.py"
name Vandan Department CE Salary 50000
name Jaydip Department IT Salary 45000
name Keval Department IT Salary 45000
PS C:\Users\Vandan\Desktop\Practical of python>
```

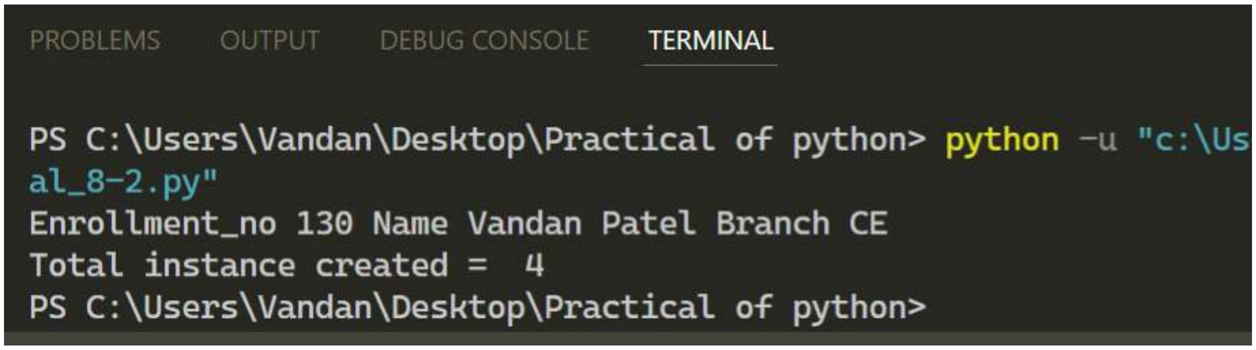
[2] Write a program to create class Student with following attributes: instance variables enrollment_no, name and branch; instance methods get_value() and print_value(); class variable cnt; static method show(). Variable cnt counts number of instances created and show() method displays value of cnt.

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",s
elf.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()
```

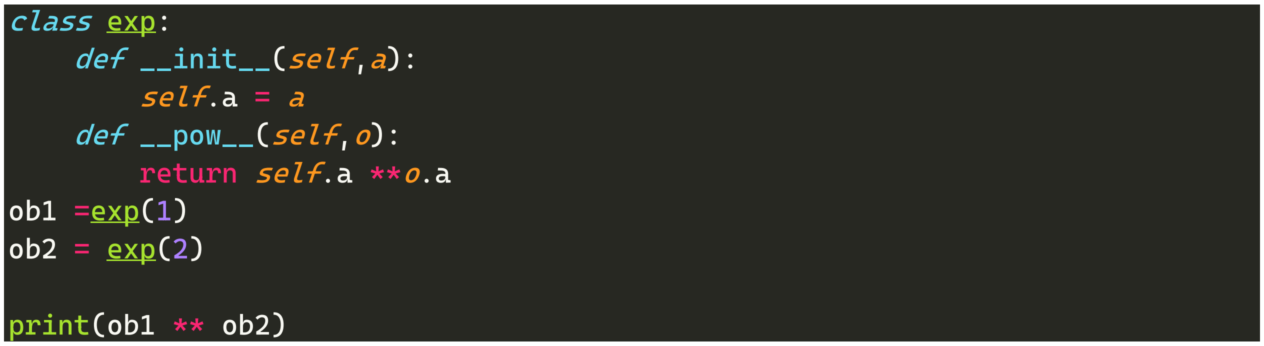
Output:



```
PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\Us
al_8-2.py"
Enrollment_no 130 Name Vandan Patel Branch CE
Total instance created =  4
PS C:\Users\Vandan\Desktop\Practical of python>
```

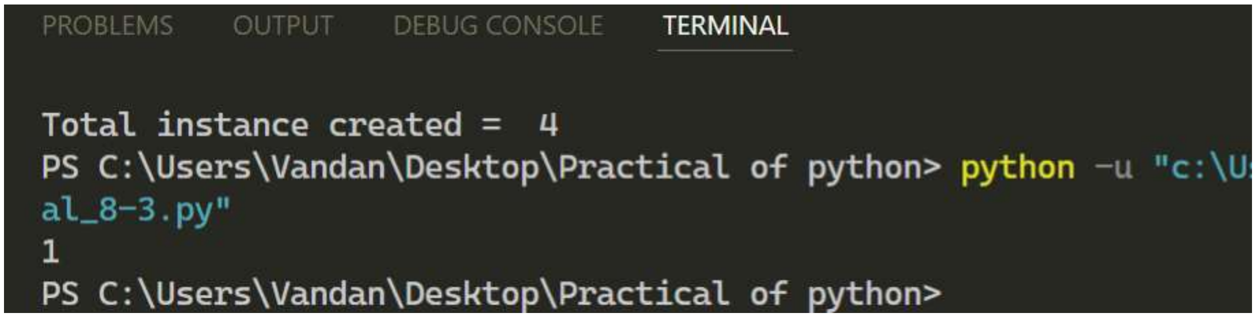
[3] Write a program to overload ** (exponential) operator.

Program :



```python
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)
```

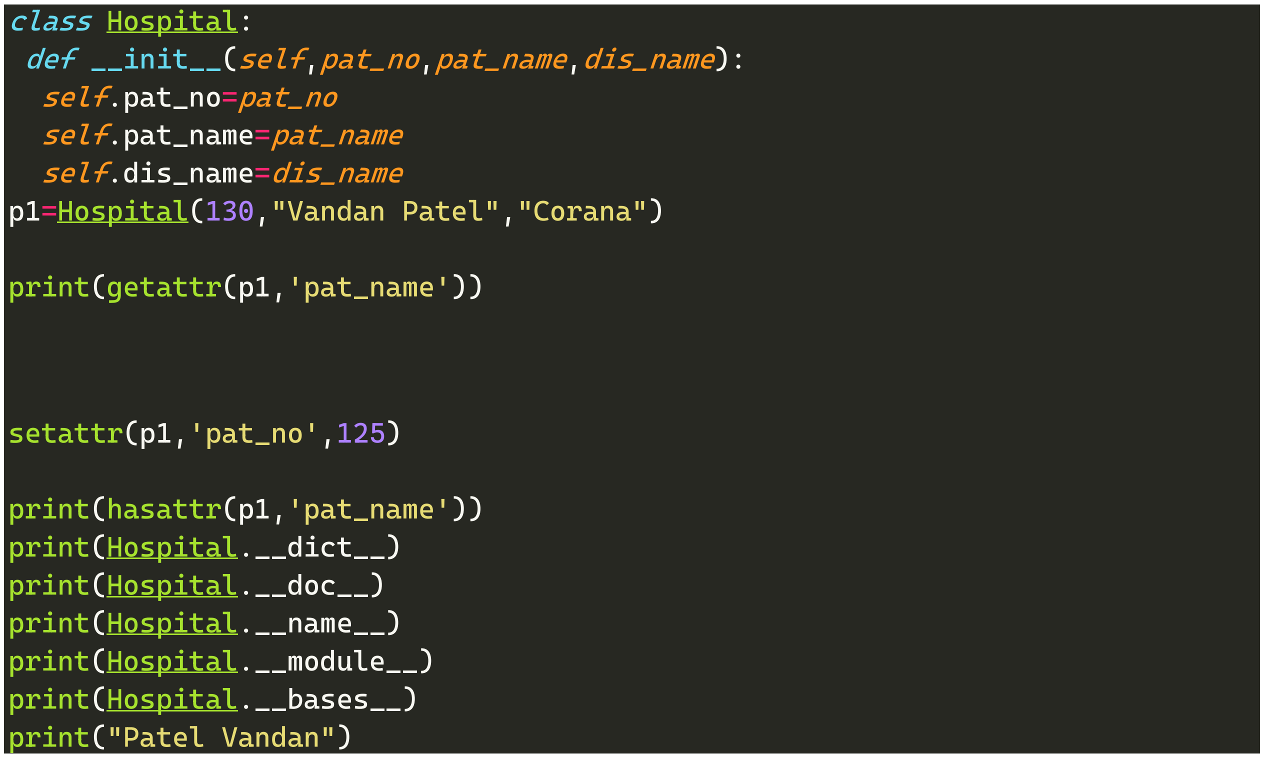
Output:


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Total instance created =  4
PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\U
al_8-3.py"
1
PS C:\Users\Vandan\Desktop\Practical of python>
```

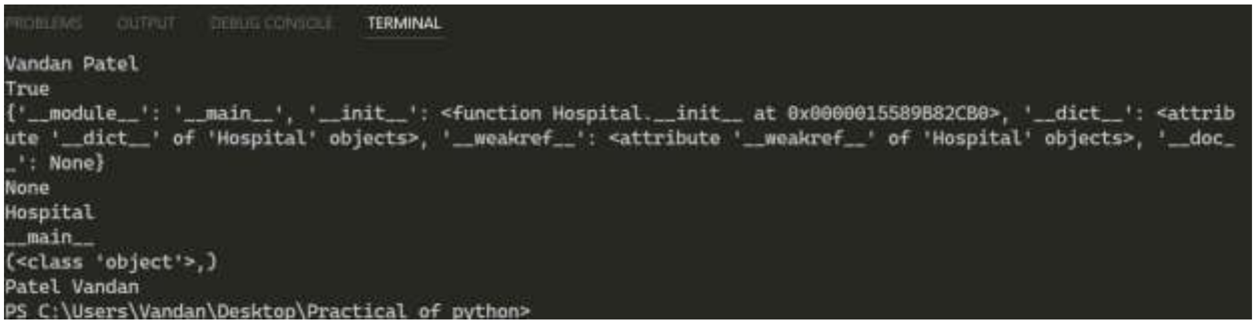
[4] Create class Hospital having attributes patient_no, patient_name and disease_name and an instance p1. Show use of methods getattr(), setattr(), delattr(), and hasattr() for p1. Display values of attributes __dict__, __doc__, __name__, __module__, __bases__ with respect to class Hospital. Delete instance p1 in the end.

Program :



```python
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:



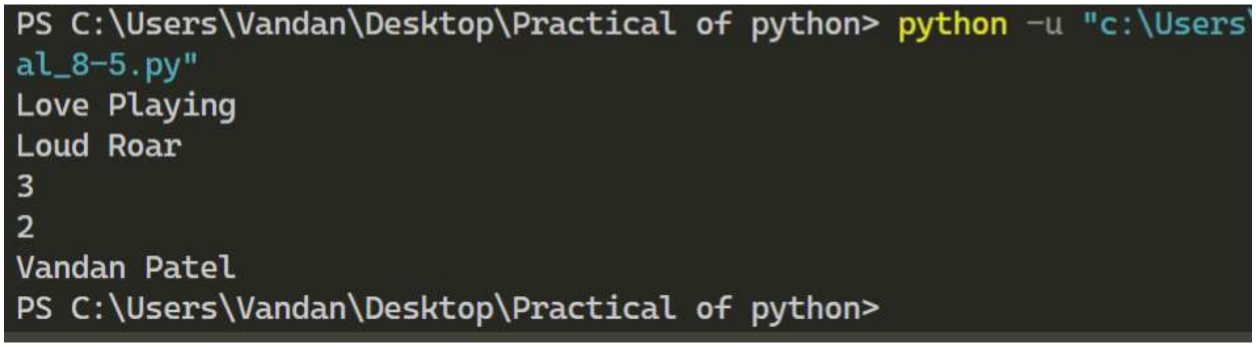
```
Vandan Patel
True
{'__module__': '__main__', '__init__': <function Hospital.__init__ at 0x0000015589B82CB0>, '__dict__': <attrib
ute '__dict__' of 'Hospital' objects>, '__weakref__': <attribute '__weakref__' of 'Hospital' objects>, '__doc_
_': None}
None
Hospital
__main__
(<class 'object'>,)
Patel Vandan
PS C:\Users\Vandan\Desktop\Practical of python>
```

[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")
```

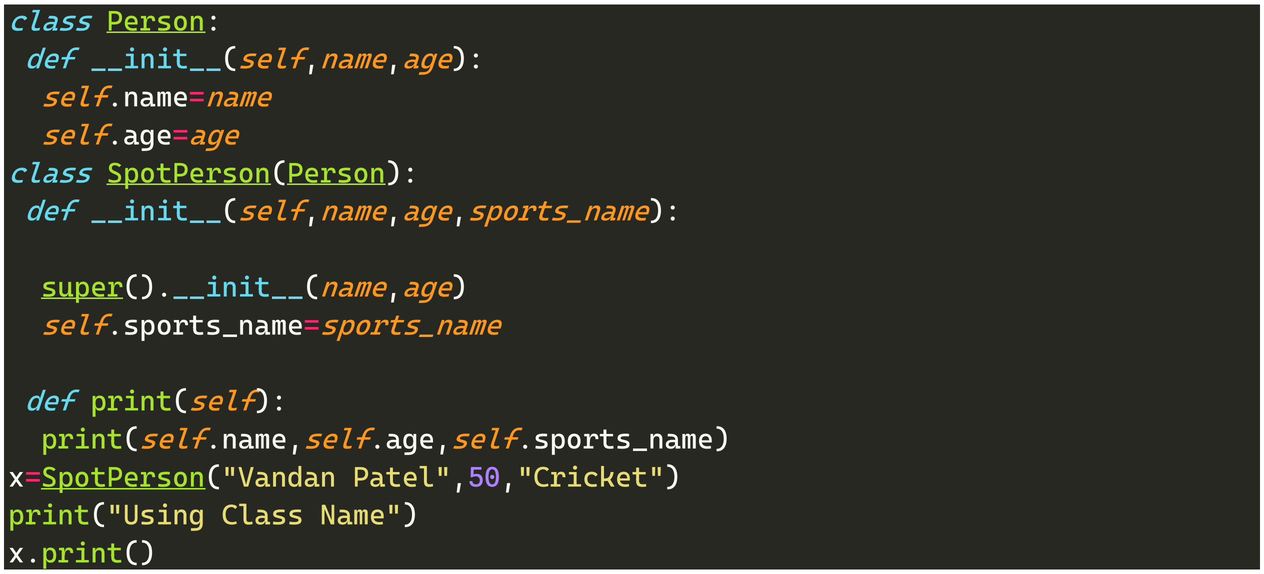
Output:



```
PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\Users
al_8-5.py"
Love Playing
Loud Roar
3
2
Vandan Patel
PS C:\Users\Vandan\Desktop\Practical of python>
```

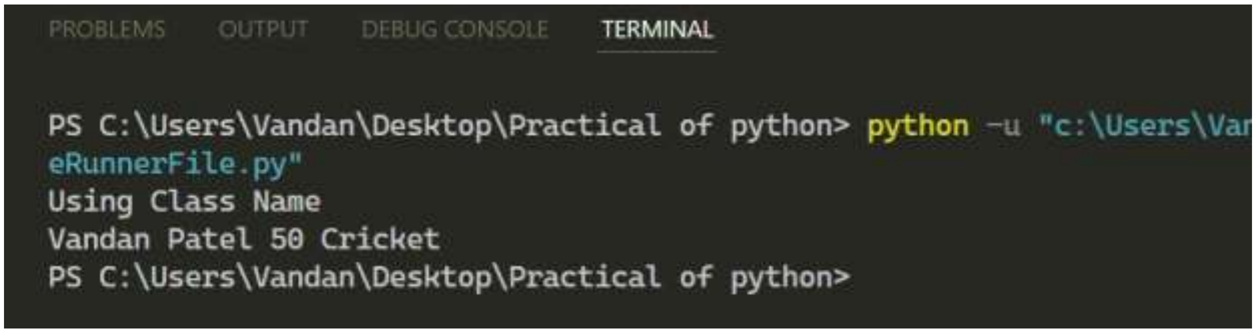
[6] Class Person with attributes- name and age is inherited by class SportPerson with attribute sport_name. Use appropriate __init__() method for both classes. Call parent __init__() method from child __init__() method with the help of (A) super() method (B) parent class name.

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()
```

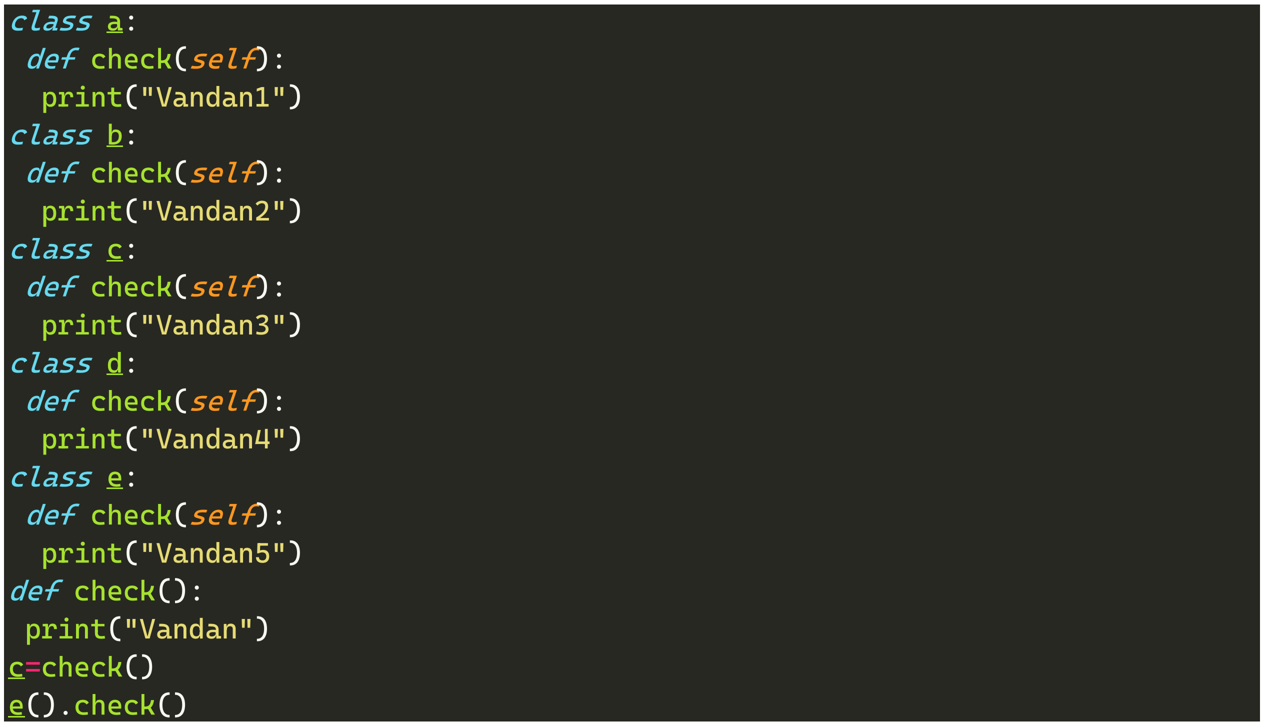
Output:


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\Users\Va
eRunnerFile.py"
Using Class Name
Vandan Patel 50 Cricket
PS C:\Users\Vandan\Desktop\Practical of python>
```

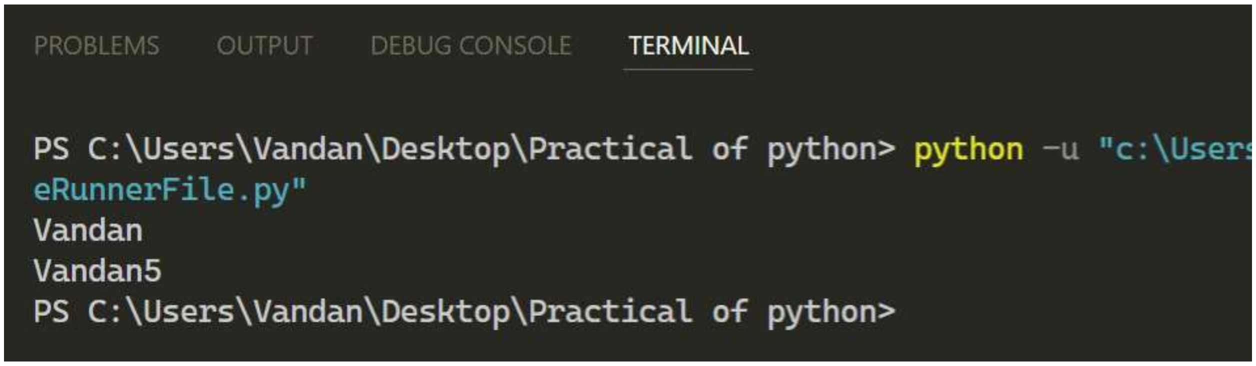
[7] Write programs to implement following scenarios where A, B, C, D, E and F are classes and check() is a method. In both scenarios, which check() method is called, when we execute statement- E().check()

Program :



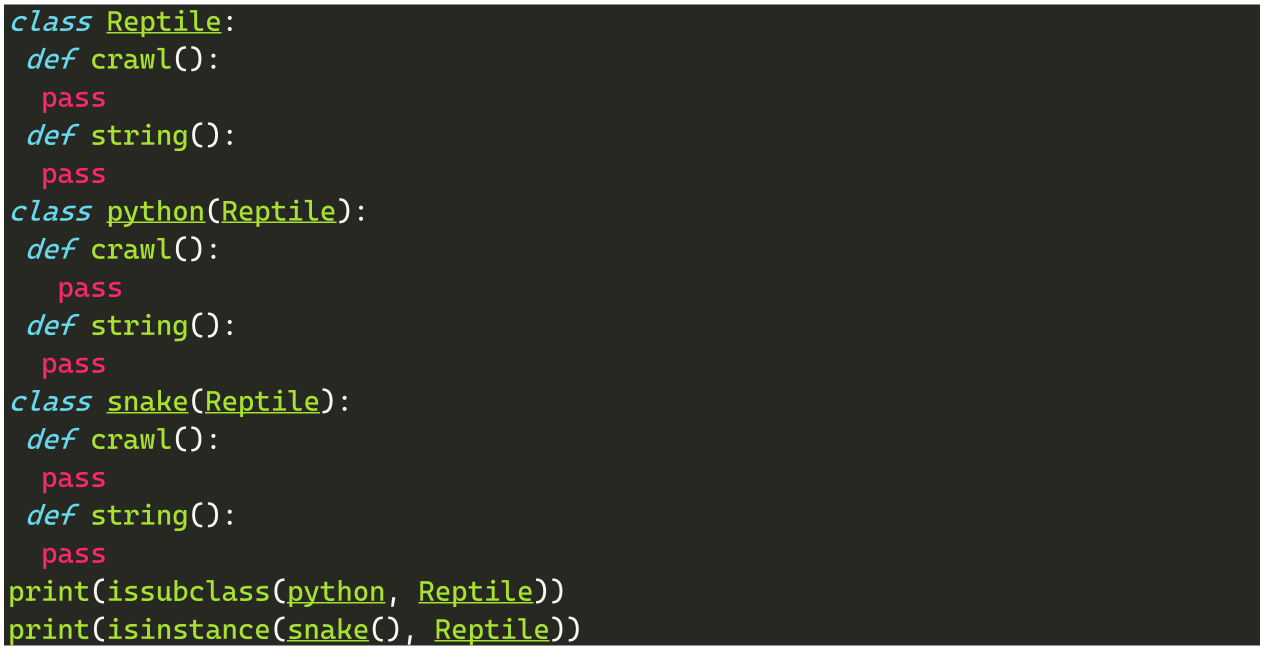
```
class a:
 def check(self):
  print("Vandan1")
class b:
 def check(self):
  print("Vandan2")
class c:
 def check(self):
  print("Vandan3")
class d:
 def check(self):
  print("Vandan4")
class e:
 def check(self):
  print("Vandan5")
def check():
 print("Vandan")
c=check()
e().check()
```

Output:



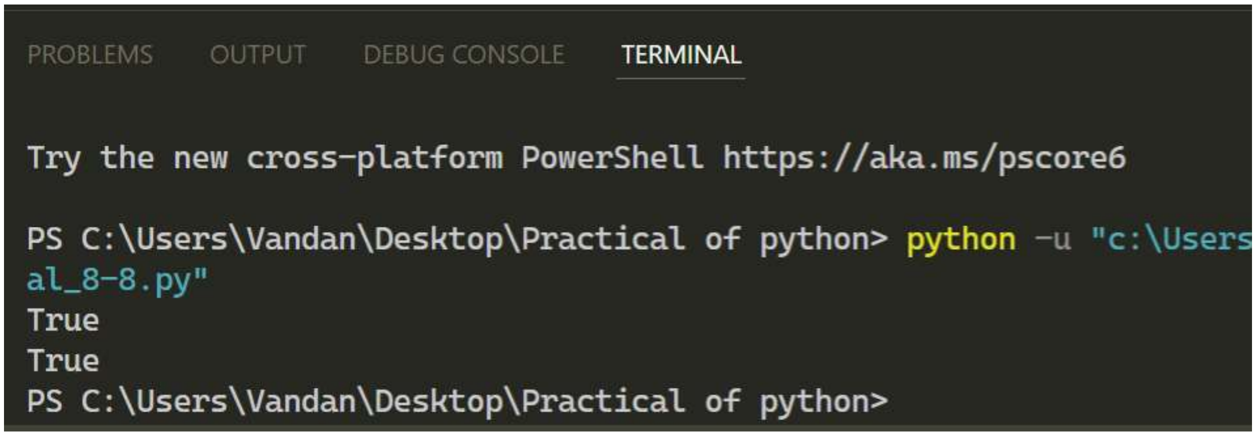
```
PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\Users
eRunnerFile.py"
Vandan
Vandan5
PS C:\Users\Vandan\Desktop\Practical of python>
```

[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:



```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Vandan\Desktop\Practical of python> python -u "c:\Users
al_8-8.py"
True
True
PS C:\Users\Vandan\Desktop\Practical of python>
```