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

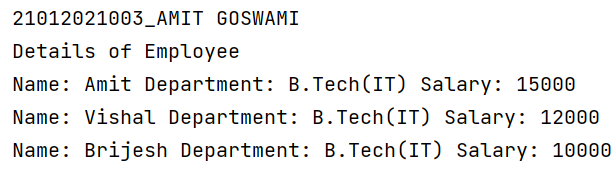
**1) Create a class Employee with data members: name, department and salary. Use constructor to initialize values and display() method for printing information of three employees.**

**Code:**

print("21012021003\_AMIT GOSWAMI")  
  
class Employee:  
 def \_\_init\_\_(self ,Name , Department, Salary):  
 self.name=Name  
 self.dept=Department  
 self.sal=Salary  
  
 def Display(self):  
 print("Name:",self.name, "Department:",self.dept , "Salary:",self.sal)

e1 = Employee("Amit", "B.Tech(IT)" ,15000)  
e2 = Employee("Vishal", "B.Tech(IT)",12000)  
e3 = Employee("Brijesh", "B.Tech(IT)",10000)  
  
print("Details of Employee")  
e1.Display()  
e2.Display()  
e3.Display()

**Output:**

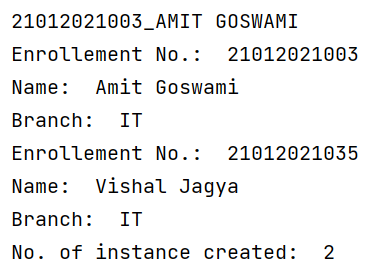
****

**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.**

**Code:**

print("21012021003\_AMIT GOSWAMI")  
class Student:  
 count = 0  
 def \_\_init\_\_(self):  
 Student.count += 1  
  
  
 def get\_value(self,Enrollment,Name, Branch):  
 self.Enrollno = Enrollment  
 self.name = Name  
 self.branch = Branch  
  
 def print\_value(self):  
 print("Enrollement No.: ", self.Enrollno)  
 print("Name: ", self.name)  
 print("Branch: ",self.branch)  
  
 def show():  
 print("No. of instance created: ", Student.count)  
  
s1 =Student()  
s1.get\_value(21012021003 , "Amit Goswami" , "IT")  
s2 = Student()  
s2.get\_value(21012021035 , "Vishal Jagya" , "IT")  
s1.print\_value()  
s2.print\_value()  
Student.show()

**Output:**

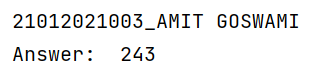
****

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

**Code:**

print("21012021003\_AMIT GOSWAMI")  
class overload:  
 def \_\_init\_\_(self,a):  
 self.a = a  
  
 def \_\_pow\_\_(self,diff):  
 return self.a\*\*diff.a  
  
e1 = overload(3)  
e2 = overload(5)  
  
x = e1\*\*e2  
  
print("Answer: ", x)

**Output:**

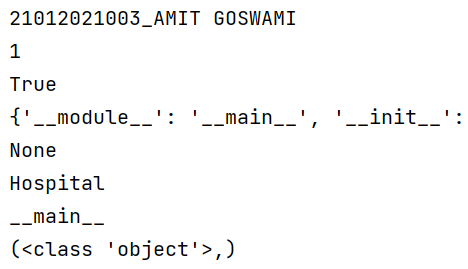
****

**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.**

**Code:**

print("21012021003\_AMIT GOSWAMI")  
class Hospital:  
 def \_\_init\_\_(self,patient\_no,patient\_name,disease\_name):  
 self.patient\_no= patient\_no  
 self.patient\_name= patient\_name  
 self.disease\_name= disease\_name  
p1 = Hospital(1,"Ram", "Fracture")  
print(getattr(p1,"patient\_no"))  
setattr(p1,"patient\_name","Raj")  
print(hasattr(p1,"patient\_name"))  
print(Hospital.\_\_dict\_\_)  
print(Hospital.\_\_doc\_\_)  
print(Hospital.\_\_name\_\_)  
print(Hospital.\_\_module\_\_)  
print(Hospital.\_\_bases\_\_)

**Output:**

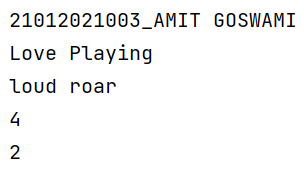
****

**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.**

**Code:**

print("21012021003\_AMIT GOSWAMI")  
  
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(4,2,"Badshah")  
c.play()  
c.roar()  
print(c.legs)  
print(c.ears)

**Output:**

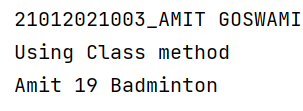
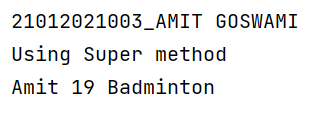
****

**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.**

**Code:**

print("21012021003\_AMIT GOSWAMI")  
class Person:  
 def \_\_init\_\_(self,name,age):  
 self.name = name  
 self.age = age  
class Sportperson(Person):  
 def \_\_init\_\_(self,name,age,Sport\_name):  
 super().\_\_init\_\_(name,age)  
 self.Sport\_name = Sport\_name  
  
  
 def print(self):  
 print(self.name,self.age,self.Sport\_name)  
x= Sportperson("Amit",19,"Badminton")  
print("Using Super method")  
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()**

Scenario-2

Scenario-1

B

C

check()

A

check()

E

D

check()

C

check()

F

check()

E

D

check()

A

check()

B

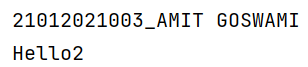
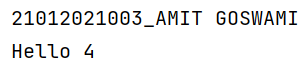
**Code:(Scenario\_1)**

print("21012021003\_AMIT GOSWAMI")  
class A:  
 def check(self):  
 print("Hello 1")  
  
class B(A):  
 pass  
  
class C(B):  
 def check(self):  
 print("Hello 3")  
  
class D(A):  
 def check(self):  
 print("Hello 4")  
  
class E(B,D):  
 pass  
  
o1 = E()  
o1.check()

**Code:(Scenario\_2)**

print("21012021003\_AMIT GOSWAMI")  
class A:  
 def check(self):  
 print("Hello1")  
  
class F:  
 def check(self):  
 print("Hello2")  
  
class B(A):  
 pass  
  
class C(F):  
 def check(self):  
 print("Hello3")  
  
class D(F):  
 def check(self):  
 print("Hello4")  
  
class E(B,C,D):  
 pass  
  
F().check()

**Output:**

****

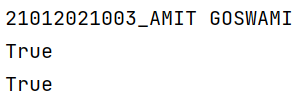
**8) Write a program in which Python and Snake sub classes implement abstract methods- crawl() and sting() of the super class Reptile. What is the output of following statements?**

**i) issubclass(Python, Reptile) ii) isinstance(Snake(), Reptile)**

**Code:**

print("21012021003\_AMIT GOSWAMI")  
class Reptile:  
 def crawl(self):  
 pass  
 def string(self):  
 pass  
class python(Reptile):  
 def crawl(self):  
 pass  
 def string(self):  
 pass  
class snake(Reptile):  
 def crawl(self):  
 pass  
 def string(self):  
 pass  
print(issubclass(python, Reptile))  
print(isinstance(snake(), Reptile))

**Output:**

****