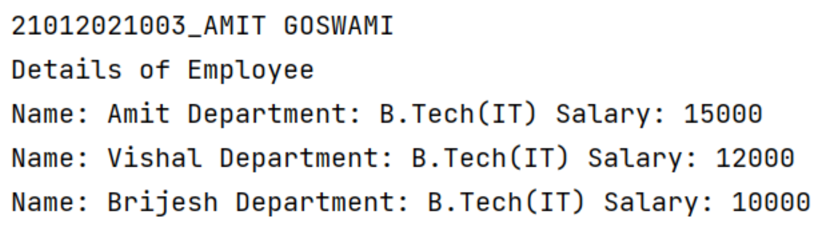
# Practical-8: Object oriented programming with python

**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:**



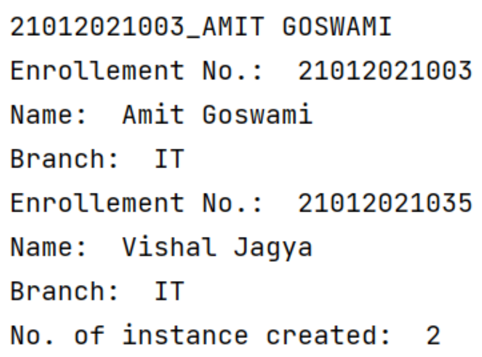
```
21012021003_AMIT GOSWAMI
Details of Employee
Name: Amit Department: B.Tech(IT) Salary: 15000
Name: Vishal Department: B.Tech(IT) Salary: 12000
Name: Brijesh Department: B.Tech(IT) Salary: 10000
```

**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:**



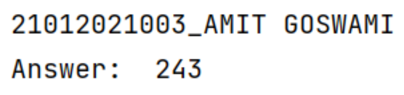
```
21012021003_AMIT GOSWAMI
Enrollement No.:  21012021003
Name:  Amit Goswami
Branch:  IT
Enrollement No.:  21012021035
Name:  Vishal Jagya
Branch:  IT
No. of instance created:  2
```

**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:**



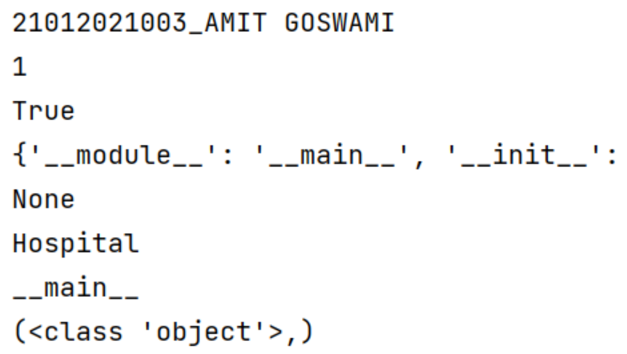
```
21012021003_AMIT GOSWAMI
Answer:  243
```

**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:**



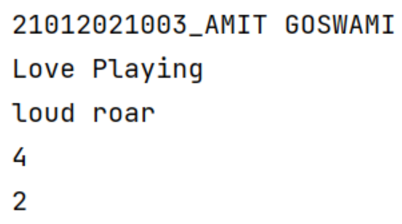
```
21012021003_AMIT GOSWAMI
1
True
{'__module__': '__main__', '__init__':
None
Hospital
__main__
(<class 'object'>,)
```

**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:**

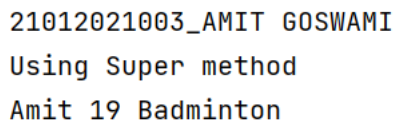

```
21012021003_AMIT GOSWAMI
Love Playing
loud roar
4
2
```

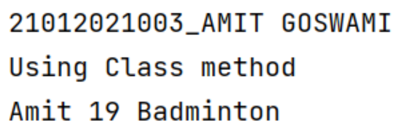
**6) Class Person with attributes- name and age is inherited by class SportPerson with attribute sport_name. Use appropriate __init__() method for both classes. Call parent __init__() method from child __init__() method with the help of (A) super() method (B) parent class name.**

**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()
```

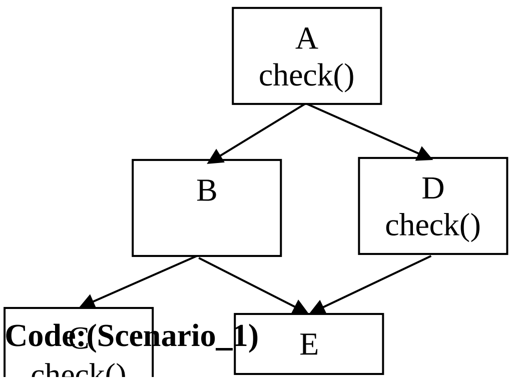
**Output:**


```
21012021003_AMIT GOSWAMI
Using Super method
Amit 19 Badminton
```

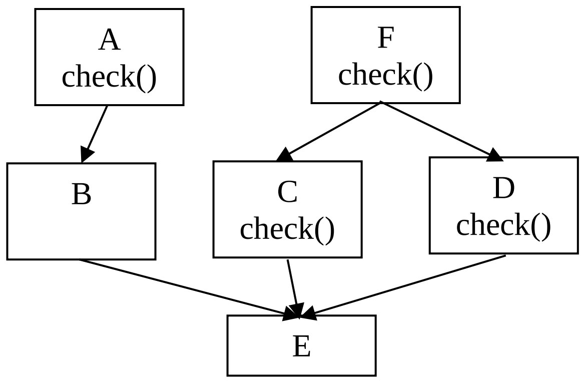

```
21012021003_AMIT GOSWAMI
Using Class method
Amit 19 Badminton
```

**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-1



Scenario-2



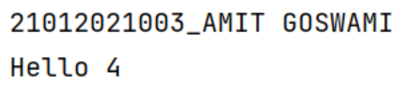
**Code:(Scenario_1)**

```
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()
```

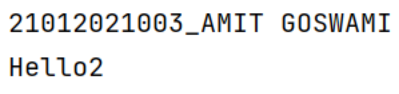
**Code:(Scenario_2)**

```
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:**



```
21012021003_AMIT GOSWAMI
Hello 4
```



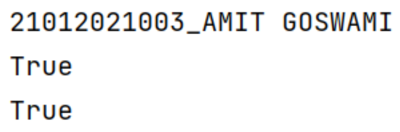
```
21012021003_AMIT GOSWAMI
Hello2
```

**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:**



```
21012021003_AMIT GOSWAMI
True
True
```