Slip no 1

1. Write a program to create a MAP with empname and deptname. Print details of all employees working in the same department, as “Mr. Joshi”.

```
class Employee(var ename:String,var dept:String)
{
def display()
{
println("---------------------------");
println("Name:"+ename);
println("Department Name:"+dept)
}
}
object Slip1
{
def main(args:Array[String])
{
val e1=new Employee("Vishnu","finance");
val e2=new Employee("Sumit","finance");
val e3= new Employee("Paresh","Marketing");
val e4 =new Employee("Tushar","Marketing");
var e5=new Employee("Akshay","Marketing");
var m1:Map[Int,Employee]=Map(1->e1,2->e2,3->e3,4->e4,5-
>e5);
for((k,v)<-m1)
{
if(v.dept.equalsIgnoreCase("marketing"))
v.display()
}
}
}
```

Slip 1

2) And **3**)=>

```
>db.student.insert({name:"Abhi",course:[{coursename:"bcs"},{coursename:"bvoc"}],marks:80,age:21,ge
nder:"male",city:"pune"})

>db.student.insert({name:"mukesh",course:[{coursename:"bcs"},{coursename:"bvoc"}],marks:60,age:22
,gender:"male",city:"pune"})

>db.student.insert({name:"manisha",course:[{coursename:"mcs"},{coursename:"bvoc"}],marks:90,age:2
2,gender:"female",city:"mumbai"})

>db.student.insert({name:"manasi",course:[{coursename:"mcs"},{coursename:"bvoc"}],marks:92,age:22
,gender:"female",city:"latur"})

>db.student.insert({name:"apurva",course:[{coursename:"mcs"},{coursename:"bvoc"}],marks:37,age:22
,gender:"female",city:"sasvad"})
```

```
>db.student.insert({name:"arati",course:[{coursename:"mcs"},{coursename:"bvoc"}],marks:32,age:22,g
ender:"female",city:"bekarai"})
```

4)

a)
```
> db.student.count({marks:{$gt:80}})
```

b)
```
> db.student.find({marks:{$lt:40}})
```

c)
```
> var my=db.student.find({marks:{$gt:70}});

> while(my.hasNext()){print(tojson(my.next()));}
```

d)
```
>db.student.find({gender:"female",$or:[{city:"pune"},{city:"mumbai"},{marks:{$lt:50}}]})
```

Slip no 2

2_1.Write a program to read five random numbers and convert it to binary and octal using user defined functions.(random number : 5M binary : 5M Octal : 5M)

```
object Slip2_1
{
def binary(num:Int)
{
var bstr=" ";//binary String
var rem=0;
println(num);
var n1=num;
while(n1>0)
{
rem=n1%2;
n1=n1/2;
bstr= rem+bstr;
}
println("Binary:"+bstr);
}
def octal(num:Int)
{
var ostr=" ";//binary String
var rem=0;
println();
println(num);
var n1=num;
while(n1>0)
{
rem=n1%8;
n1=n1/8;
ostr= rem+ostr;
}
println("octal:"+ostr);
}
def main(args:Array[String])
{
val r=new scala.util.Random;
binary(r.nextInt(15))
octal(r.nextInt(15))
}
}
```

2_2.Write a program to calculate average of all prime numbers between n1 and n2 (take n1 and n2 from user).(accept n1, n2 : 5M prime numbers : 5M average :5M)

```
object Slip2_2
```

```
{
def main(args:Array[String])
{
var n1=0;
var n2=0;
var count=0;
var pcount=0;
var sum=0;
var prime=" ";
println("Enter two numbers:");
n1=scala.io.StdIn.readInt();
n2=scala.io.StdIn.readInt();
for(i<-n1 to n2)
{
count=0;
for(j<-1 to i )
{
if(i%j==0)
{
count=count+1;
}
}
if(count==2)
{
prime=prime+" "+i;
pcount=pcount+1;
sum=sum+i;
}
}
println("prime numbers:"+prime);
println("average:"+sum/pcount);
}
}
```

Slip 2

2)

```
> db.product.insert({name:"robot",price:12000})

> db.product.insert({name:"toycar",price:2000})

> db.product.insert({name:"cricketset",price:9000})

> db.product.insert({name:"studymaterial",price:19000})
```

3)

```
>db.order.insert({orderno:3736,custName:"arunkumar",product:{productName:"toycar",price:20000},o
rder_date:"12/2/2019",stetus:"processed",Totalbill:2039,invoice:{invoiceNO:67564,bill:2039,date:"17/2
/2019"}})
```

```
>db.order.insert({orderno:3737,custName:"arunkumar",product:{productName:"robot",price:12000},or
der_date:"11/3/2019",stetus:"processed",Totalbill:12800,invoice:{invoiceNO:67574,bill:12039,date:"17/
3/2019"}})

>db.order.insert({orderno:3738,custName:"arunkumar",product:{productName:"cricketset",price:9000}
,order
_date:"15/5/2019",stetus:"in process",Totalbill:9050})

>db.order.insert({orderno:3739,custName:"mukeshpatil",product:{productName:"studentmaterial",pric
e:19000},order_date:"15/8/2019",stetus:"in process",Totalbill:19080})
```

4)

a)> db.product.find().pretty()

b) > db.order.find({Totalbill:{$lt:10000}})

c) > db.order.find({stetus:"in process"})

d) >db.order.find({custName:"arunkumar",stetus:"processed"})

Slip no 3

3.Create an abstract class Order (id, description). Derive two classes Purchase Order and Sales Order with details of Supplier and Customer respectively. Create object of each Purchase Order And Sales Order. Display the details of the supplier and customer.

```
abstract class Order()
{
var orderid:Int=0
var odescription:String=" ";
}
class PurchaseOrder( var oid:Int,val descrip:String,var sid:Int,var
sname:String,var pno:Long) extends Order()
{
orderid=oid;
odescription=descrip;
def display()
{
println("Order Id:"+orderid);
println("Description:"+odescription);
println("Supplier Id:"+sid);
println("Supplier Name:"+sname);
println("Phone Number:"+pno);
}
}
class SalesOrder(var oid:Int,val descrip:String,var cid:Int,var
cname:String,var pno:Long) extends Order()
{
orderid=oid;
odescription=descrip;
def display()
{
println("Order Id:"+orderid);
println("Description:"+odescription);
println("Customer Id:"+cid);
println("Customer Name:"+cname);
println("Phone Number:"+pno);
}
}
object Slip3 {
def main(args:Array[String])
{
var c1=new SalesOrder(1,"Two
Laptops",200,"XYZ",233221);
var s1=new PurchaseOrder(2,"Three
Computers",101,"ABC",211231);
println("Purchase Order");
println("--------------------------------------------");
c1.display();
println("Sales Orders");
println("--------------------------------------------");
s1.display();
}
}
```

slip 3

2)

```
> db.book.insert({BName:"shyamchiaai",cost:700,author:"sane guruji",published:2007})

> db.book.insert({BName:"Two Saints",cost:1700,author:"raguramkrishna",published:2017})

> db.book.insert({BName:"ramkrushna paramhans",cost:800,author:"raguramkrishna",published: 2017})

>db.book.insert({BName:"DMS",cost:300,author:"raguramkrishna",published:2005})
```

3)

```
> db.publisher.insert({pname:"OReilly",language:"English",books:[{BName:"ramkrushna
paramhans"},{BName:"Two Saints"}],city:"mumbai"})

>db.publisher.insert({pname:"vision",language:"English",books:[{BName:"DMS"}],city:"pune"})

> db.publisher.insert({pname:"OReilly",language:"marathi",books:[{BName:"shyamchi
aai"}],city:"mumbai"})
```

4)

a)> db.publisher.find({city:"mumbai"})

b)> db.book.find({cost:{$lt:1000}})

c) > db.book.find({author:"raguramkrishna",published:2017})

d)> > db.publisher.find({pname:"OReilly",$or:[{language:"English"},{language:"marathi"}]})

Slip no 4

4.Write a program to calculate transpose of a matrix and check if there sultant matrix is lower triangular or not.(accept : 5 M transpose : 10M check lower triangular:10M display:5M)

```
object Slip4
{
def main(args:Array[String])
{
var mat=Array.ofDim[Int](3,3);
var rmat=Array.ofDim[Int](3,3);
var isLower:Boolean=true;
println("Enter Matrix");
for(i<-0 to 2)
{
for(j<-0 to 2)
{
mat(i)(j)=scala.io.StdIn.readInt();
}
}
println("Matrix is:");
for(i<-0 to 2)
{
for(j<-0 to 2)
{
print(mat(i)(j)+" ");
}
println();
}
for(i<-0 to 2)
{
for(j<-0 to 2)
{
rmat(i)(j)=mat(j)(i);
}
}
println("Transepose of Matrix is:");
for(i<-0 to 2)
{
for(j<-0 to 2)
{
print(rmat(i)(j)+" ");
}
println();
}
for(i<-0 to 2)
{
for(j<-0 to 2)
{
if(i<j)
{
if(rmat(i)(j)!=0)
isLower=false;
}
}
}
if(isLower==true)
println("Is Lower Triangular");
else
println("Is not Lower Triangular");
}
}
```

slip 4

2)

```
>db.Hospital.insert({Hno:1,Hname:"AAA",Specialization:["Pediatric","Gynaec","Orthopaedic"],People:[{Pname:"PQR",Rating:4},{Pname:"SDE",Rating:5}],Doctor:[{"Dname" : "WWW",
"Visit" : "Sunday"}]})

>db.Hospital.insert({Hno:2,Hname:"BBB",Specialization:["Gynaec","Orthopaedic"],People:[{Pname:"POP",Rating:2},{Pname:"SDE",Rating:3}],Doctor:[{"Dname":"XXX",Visit:"Monday"}]})

>db.Hospital.insert({Hno:3,Hname:"CCC",Specialization:["Gynaec","Orthopaedic","Pediatric"],People:[{Pname:"KLO",Rating:3},{Pname:"LPO",Rating:3}],Doctor:[{"Dname" : "XXX","Visit":"Tuesday"}]})
```

4)

a) > db.Hospital.find({Specialization:"Pediatric"})

b)>db.Hospital.find({Hname:"CCC","Doctor.Visit":"Tuesday"})

c)>db.Hospital.find({Specialization:{$not:{$size:1}},"Doctor.Dname":"XXX"})

d) > db.Hospital.find({"People.Rating":{ $gt: 3 },Hname:"AAA"})

Slip no 5

5.Write a program to create two sets of strings and find common strings between them. Merge sets after removing common strings. Display resultant set.(create sets:10M find common elements:5M merge removing common :1

```
object Slip5 {
def main(args:Array[String]) {
var str1:Set[String]=Set("Hello","good","Morning");
var str2:Set[String]=Set("Hello","good","night");
var str3=str1.diff(str2);
println(str1);
println(str2);
println(str3);
var str4=str2.diff(str1);
println(str4);
str3++=str4;
println(str3)
}
}
```

slip 5 2) & 3)

```
>db.post.insert({title:"online",url:"www.abc.com",tag:["food","travel"],pname:"mukesh",pdate:new Date("2019-03-12"),like:89,user:[{name:"abhi",comment:"good",message:"do best", cdate:new Date("2020-03-12"),like:1}]})

>db.post.insert({title:"wetpet",url:"www.wetpet.com",tag:["food","travel",],pname:"Amit",pdate:new Date("2018-03-12"),like:82,user:[{name:"abhi",comment:"good",message:"do best",time:"4pm",like:1},{name:"mukesh",comment:"best",message:"success", cdate:new Date("2008-11-12"),like:2}]})

>db.post.insert({title:"wetpet",url:"www.wetpet.com",tag:["food","travel","magic"],pname:"abhijeet",pdate:newDate("2017-03-12"),like:182,user:[{name:"sagar",comment:"like",message:" dobest",time:"4pm",like:1},{name:"mukesh",comment:"best",message:"success", cdate:new Date("2019-03-12"),like:2}]})

>db.post.insert({title:"nonveg",url:"www.non.com",tag:["food","travel","chiken"],pname:"Amit",pdate: new Date("2019-07-12"),like:82,user:[{name:"manisha",comment:"good",message:"dobest",time:"4pm",like:0},{name:"manasi",comment:"best",message:"success", cdate:new Date("2018-03-12"),like:0}]})
```

4)

a) >db.post.find({tag:"food"})

b) >db.post.find({pname:"Amit"})

c) > db.post.find({tag:"travel",pdate:{"$lte":new Date("2018-03-11")},"user.name":"sagar","user.comment":"like"})

d) > db.post.find({$or:[{"user.cdate":{$lte:new Date("2019-08-07")}},{"user.like":0}]})

Slip no 6

6.Write a program to read a character and a string from user and remove first and last occurrence of the character from the string. Display resultant string after reversing its case.

```
object Slip6
{
def reverseString(ch:Char):Char=
{
if(ch.isLower)
ch.toUpper;
else
ch.toLower;
}
def main(args:Array[String])
{
var ch=' ';
var str=" ";
println("Enter String:");
str=scala.io.StdIn.readLine();
var str1=new StringBuilder(str);
println("Enter character:");
ch=scala.io.StdIn.readChar();
str1.deleteCharAt(str1.indexOf(ch.toString()));
var
str3=str1.deleteCharAt(str1.lastIndexOf(ch.toString())).toString;
var str4=str3.map(reverseString)
println(str4);
}
}
```

slip6

2)

```
> db.turisum.insert({name:"veenaword",rate:9,package:[{pname:"shillong",cost:10000},{pnam
e:"gujart",cost:7000},{pname:"karnataka",cost:6000}]})

>db.turisum.insert({name:"rohit",rate:7,package:[{pname:"shillong",cost:10000},{pname:"rujan",cost:70
00}]})
```

3)

```
>db.tour.insert({sourc:"john",destination:"shillong",toerisumName:"veenaword",tourisumrate:8000,exp
ense:20000,year:2018,customer:[{cname:"mukesh",city:"pune"},{cname:"abhijeet
sangita",city:"baramati"},{cname:"manisha",city:"15no"},{cname:"manasi",city:"latur"}]})

>db.tour.insert({sourc:"john",destination:"karnataka",toerisumName:"veena
word",tourisumrate:80090,expense:20900,year:2017,customer:[{cname:"mukesh",city:"pune"},{cname:
"abhijeetsangita",city:"baramati"},{cname:"manisha",city:"15no"},{cname:"manasi",city:"latur"}]})
```

```
>db.tour.insert({sourc:"john",destination:"rajasthan",toerisumName:"rohit",tourisumrate:6000,expense
:30400,year:2019,customer:[{cname:"mukesh",city:"pune"},{cname:"abhijeet
sangita",city:"baramati"},{cname:"manisha",city:"15no"},{cname:"manasi",city:"latur"}]})
```

```
>db.tour.insert({sourc:"john",destination:"taj",toerisumName:"rohit",tourisumrate:60090,expense:1040
0,year:2016,customer:[{cname:"mukesh",city:"pune"},{cname:"abhijeetsangita",city:"baramati"},{cnam
e:"manisha",city:"15no"},{cname:"manasi",city:"latur"}]})
```

4)

a) >db.turisum.find({name:"veena word"}).pretty()

b) >db.turisum.find({}).sort({"rate":-1}).limit(1)

c)>db.tour.aggregate([{"$sort":{"year":1}},{"$limit":3},{$group:{_id:null,"count":{"$sum":"$expense"}}}])

d) > db.tour.find({destination:"shillong"})

Slip no 7

slip 7

2) & **3**)

> db.scien.insert({fname:"mukesh",lname:"navse",BOD:newDate("1952-04-18"),DOD:"still alive",field:["tcs","java","c","sql"],award:[{name:"turingmachine",year:1976},{name:"robotic",year:1998 },{name:"codetalent",year:1995}]})

> db.scien.insert({fname:"abhi",lname:"nalave",BOD:newDate("1972-04-18"),DOD:"still alive",field:["tcs","java","sql"],award:[{name:"codemaster",year:1976},{name:"robot",year:1998},{name: "puzzletalent",year:1995}]})

>db.scien.insert({fname:"manisha",lname:"hipparkar",BOD:new Date("1942-04-18"),DOD:new Date("2009-08-06"),field:["tcs","java"],award:[{name:"topper",year:1976},{n ame:"puraskar",year:1998},{name:"puzzletalent",year:1995}]})

4)

a) > db.scien.find({ lname: { $regex: /n/ } })

b) > db.scien.find({BOD:{"$gt":new Date("1950-03-11")},DOD:"still alive"})

c)>db.scien.aggregate([{$group:{_id:{year:"$award.year",Name:"$award.name"}}}])

d) > db.scien.find({"award.name":"turingmachine","award.year":{$lt:1980},field:{$size:4}})

Slip no 8

8.Create array of strings and read a new string from user. Display all the strings from the array that contain the new string.

```
object Slip8
{
def main(args:Array[String])
{
var str:Array[String]=Array("Hello Good Morning","Hello
Good Night","Hello Good Afternoon");
var str1=" ";
println("Enter string:");
str1=scala.io.StdIn.readLine();
var str2=str :+str1;
for(j<-str2)
{
println(j);
}
}
}
```

slip8

2) & **3**)

```
>db.item.insert({itemName:"planner",tag:["wash","food","vehicle"],status:"A",height:5,width:9,instack:
15,warehouse:[{location:"pune",quntity:36},{location:"mumbai",quntity:67}]})

>db.item.insert({itemName:"toycar",tag:["food","vehicle"],status:"D",height:5,width:9,instack:15,wareh
ouse:[{location:"pune",quntity:36},{location:"mumbai",quntity:67}]})

> db.item.insert({itemName:"roboticcar",tag:["food","vehicle"],status:"A",height:9,width:9,instack
:5,warehouse:[{location:"pune",quntity:26},{location:"mumbai",quntity:17}]})

>db.item.insert({itemName:"bag",tag:["food","vehicle","school","travel"],status:"c",height:19,width:39,i
nstack:75,warehouse:[{location:"surat",quntity:26},{location:"lanavala",quntity:17}]})
```

4)

a) `> db.item.find({status:"D","warehouse.quntity":{$gt:30}})`

b) `> db.item.find({"tag":{$size:3}})`

c) `>db.item.find({$or:[{status:"A"},{"warehouse.quntity":{$lt:30},height:{$gt:10}}]})`

d) `> db.item.find({itemName:"planner",instack:{$lt:20}})`

Slip no 9

9.Create a MAP for storing the following information about 5 students, where each Student is described with Name and percentage. Display Student information with highest percentage.

```
class Student(var rno:Int,var sname:String,var sub1:Int,var sub2:Int)
{
var ptage:Float=(sub1+sub2)/2;
def display()
{
println("Roll No:"+rno);
println("Name:"+sname);
println("Percentage:"+ptage);
}
}
object Slip9
{
def main(args:Array[String])
{
val s1=new Student(1,"Akshay Borse",80,70);
val s2=new Student(2,"Sumit Amritkar",75,85);
val s3=new Student(3,"Vishnu Khatale",77,87);
val s4=new Student(4,"Aniket Borse",89,99);
val s5=new Student(5,"Tushar Amrutkar",84,87);
val m1:Map[Int,Student]=Map(1->s1,2->s2,3->s3,4->s4,5->s5);
var max=m1(1).ptage;
for((k,v)<-m1)
{
if(m1(k).ptage>max)
max=m1(k).ptage;
}
for((k,v)<-m1)
{
if(m1(k).ptage==max)
m1(k).display()
}
}
}
```

slip 9

2) & **3**)

```
>db.transaction.insert({itemName:"toy",customerName:"john",paymentmode:"debitcard",payment:800
0})

>db.transaction.insert({itemName:"car",customerName:"john",paymentmode:"creditcard",payment:40
00})

>db.transaction.insert({itemName:"bag",customerName:"mukesh",paymentmode:"cash",payment:5000
})
```

```
>db.transaction.insert({itemName:"airlineticket",customerName:"rohit",paymentmode:"cash",payment:
50090})
```

```
>db.transaction.insert({itemName:"mango",customerName:"abhijeet",paymentmode:"creditcard",pay
ment:8000})
```

```
>db.transaction.insert({itemName:"bus",customerName:"manasi",paymentmode:"debitcard",payment:
7000})
```

4)
a) `> db.transaction.find({customerName:"john"})`

b) `> db.transaction.find({paymentmode:"debitcard"})`

c)
```
>db.transaction.aggregate([{$match:{"paymentmode":"creditcard"}},{$group:{_id:null,"count":{"$sum
":"$payment"}}}])
```

d)
```
>db.transaction.aggregate([{$group:{_id:"$paymentmode","count":{"$sum":"$payment"}}}])
```

Slip no 10

10.Create abstract class Shape with abstract functions volume() and display().Extend two classes Cube and Cylinder from it. Create object of Cube and Cylinder, Calculate volume of each and display it.

```
abstract class Shape
{
def volume():Double;
def display();
}
class Cylinder(var r:Int,var h:Int) extends Shape
{
def volume():Double=
{
return 3.14*r*r*h;
}
def display()
{
println("Volume Cylinder :"+volume());
}
}
class Cube(var s:Int) extends Shape
{
def volume():Double=
{
return s*s*s;
}
def display()
{
println("Volume of cube:"+volume());
}
}
object Slip10
{
def main(args:Array[String])
{
val cyl=new Cylinder(1,1);
cyl.display();
val cub=new Cube(3);
cub.display();
}
}
```

slip10

2)

```
>db.custome.insert({cname:"mukesh",modelname:"samsungj6",amount:20000})

>db.custome.insert({cname:"abhijeet",modelname:"samsungj6",amount:20060})

> db.custome.insert({cname:"manasi",modelname:"iphone7+",amount:30060})
> db.custome.insert({cname:"manisha",modelname:"iphone7+",amount:30070})

> db.custome.insert({cname:"dipak",modelname:"iphone7+",amount:30800})
```

3)

```
>db.shopping.insert({brandname:"samsung",rate:6,model:[{mname:"s40",ram:"3GB",rom:"32GB",rate:4
},{mname:"j6",ram:"4GB",rom:"32GB",rate:7},{mname:"j7",ram:"6GB",rom:"64GB",rate:6}]})

>db.shopping.insert({brandname:"vivo",rate:8,model:[{mname:"Y55",ram:"3GB",rom:"32GB",rate:6},{m
name:"Ys5",ram:"4GB",rom:"32GB",rate:4},{mname:"YYY",ram:"6GB",rom:"64GB",rate:6}]})
```

4)

a) `>db.shopping.find({"model.ram":"3GB","model.rom":"32GB"})`

b) `> db.custome.find({modelname:"samsung j6"})`

c) `> db.shopping.aggregate([{"$sort":{"rate":-1}},{"$limit":1},{$group:{_id:"$brandname"}}])`

d) `> db.custome.find().sort( { "cname": 1 } )`

Silp no 11

2 Model the following Society relations between people working in “HCL”, as a graph model, and answer the queries using Cypher. A person can be a friend of another person. A person may have siblings (brothers / sisters), A person may be a parent (mother/father) of another person. A person stays either

create(p:Person{name:"Tushar",age:24}) return p;

create(p:Person{name:"Ganesh",age:45}) return p;

create(p:Person{name:"Amar",age:26}) return p;

create(b:Brother{name:"Vinay"}) return b;

create(s:Sister{name:"Sanskruti"}) return s;

create(c:Children{name:"Amit"}) return c;

create(l:Location{name:"Pune"}) return l;

create(l:Location{name:"Mumbai"}) return l;

create(l:Location{name:"Kolhapur"}) return l;

match(p:Person),(p1:Person) where p.name="Tushar" and p1.name="Ganesh" create (p)-[:Friend_of]->(p1) return p,p1;

match(p:Person),(b:Brother) where p.name="Ganesh" and b.name="Vinay" create (p)-[:Brother_of]->(b) return p,b;

match(p:Person),(c:Children) where p.name="Ganesh" and c.name="Amit" create (p)-[:Parent_of]->(c) return p,c;

match(p:Person),(l:Location) where p.name="Ganesh" and l.name="Pune" create (p)-[:Stays_in]->(l) return p,l;

match(p:Person),(l:Location) where p.name="Tushar" and l.name="Mumbai" create (p)-[:Stays_in]->(l) return p,l;

match(p:Person),(l:Location) where p.name="Amar" and l.name="Mumbai" create (p)-[:Stays_in]->(l) return p,l;

Queries 1) Display the names of people living in Mumbai.

match(p:Person),(l:Location) where l.name="Mumbai" and (p)-[:Stays_in]->(l) return p.name;

2) Display the nodes having age above 40.

```
match(p:Person) where p.age>40 return p.name;
```

Slip no 12

12.Write a program for multiplication of two matrices. Find determinant ofresultant matrix.

```
object Slip12
{
def main(args:Array[String])
{
val arr1=Array.ofDim[Int](2,2);//1st array
val arr2=Array.ofDim[Int](2,2);//2nd array
var rarry=Array.ofDim[Int](2,2)//resultant Array
println("Enter Matrix1");
for(i<-0 to 1)
{
for(j<-0 to 1)
{
arr1(i)(j)=scala.io.StdIn.readInt();//read Array1 element
}
}
println("Enter Matrix2");
for(i<-0 to 1)
{
for(j<-0 to 1)
{
arr2(i)(j)=scala.io.StdIn.readInt();//read Array2 element
}
}
println("MATRIX -1");
for(i<-0 to 1)
{
for(j<-0 to 1)
{
print(arr1(i)(j)+" ");//print Array
Element
}
println();
}
println("MATRIX -2");
for(i<-0 to 1)
{
for(j<-0 to 1)
{
print(arr2(i)(j)+" ");//print Array
Element
}
println();
}
for(i<-0 to 1)
{
for(j<-0 to 1)
{
rarry(i)(j)=0;
for(k<-0 to 1)
rarry(i)(j)=rarry(i)(j)+arr1(i)(k)*arr2(k)(j);//multiplication
}
}
println("RESULTANT MATRIX");
for(i<-0 to 1)
{
for(j<-0 to 1)
{
print(rarry(i)(j)+" ");//print Array
Element
}
println();
}
var det=(rarry(0)(0)*rarry(1)(1))-
(rarry(0)(1)*rarry(1)(0));
println("Determinant:"+det);
}
}
```

2)Model the following Dairy Brand information as a graph model , and answer the following queries using Cypher. There are various dairy brands like Amul, Go, Britannia, Gokul etc. Their popularity varies across different states in India. The popularity is measured as %, with a high popularity defined as >=90%, Medium Popularity between 50 to 90%, and Low popularity <50%. Each brand manufactures various types of Dairy products like milk, butter, cheese, Curd etc. The milk product can be categorized into Low fat/medium fat or high fat content type.

```
create(b:Brands{name:"Gokul",popularity:"95%"}) return b;

create(b:Brands{name:"Britannia",popularity:"45%"}) return b;

create(b:Brands{name:"Amul",popularity:"75%"}) return b;

create(p:Products{name:"Cheese"}) return p;

create(p:Products{name:"Curd"}) return p;

create(p:Products{name:"Milk"}) return p;

create(p:Products{name:"Butter"}) return p;

create(c:Category{name:"Low Fat"}) return c;

create(c:Category{name:"Medium Fat"}) return c;
```

create(c:Category{name:"Medium Fat"}) return c;

match(b:Brands),(p:Products) where b.name="Amul" and p.name="Cheese" create (b)-[:Has_Products]->(p) return b,p;

match(b:Brands),(p:Products) where b.name="Gokul" and p.name="Milk" create (b)-[:Has_Products]->(p) return b,p;

match(p:Products),(c:Category) where p.name="Curd" and c.name="Medium Fat" create (p)-[:Has_Category]->(c) return p,c;

match(p:Products),(c:Category) where p.name="Milk" and c.name="Low Fat" create (p)-[:Has_Category]->(c) return p,c;

match(p:Products),(c:Category) where p.name="Cheese" and c.name="High Fat" create (p)-[:Has_Category]->(c) return p,c;

Queries

1}List the names of different brands considered in your graph.

match(b:Brands) return *;

2) List the brands that are highly popular in Maharashtra.

match(b:Brands) where b.popularity>="90%" return b;

Slip no 13

13.Write a program to merge two sets of integers and calculate sum of all integers in the merged set. Also display largest and smallest element from merged set.

```
import scala.collection.mutable.Set
object Slip13
{
def main(args:Array[String])
{
var s1=Set(1,2,3,4,5,6);
var s2=Set(4,5,6,7,8);
s1++=s2;
println(s1);
println("Sum:"+s1.sum);
println("Maximum:"+s1.max);
println("Minimum:"+s1.min);
}
}
```

2 Model the following Import Export information as a graph model, and answer the following queries using Cypher. There are countries which import and export products to each other. Products are produced across different states in a country. Production of the products is measured in %. A product can be exported if its production exceeds 60%. A product needs to be imported if its consumption percentage is more than its production percentage in a country.

```
create (c:Country{name:"India"}) return c;

create (c:Country{name:"USA"}) return c;

create (c:Country{name:"Israil"}) return c;

create (c:Country{name:"Arab"}) return c;

create (c:Country{name:"Europe"}) return c;

create (s:States{name:"Maharashtra"}) return s;

create (s:States{name:"Punjab"}) return s;

create (s:States{name:"California"}) return s;

create (p:Product{name:"Wheat",production:"75%"}) return p;

create (p:Product{name:"Oil",production:"90%"}) return p;

create (p:Product{name:"Sugar",production:"50%"}) return p;

create (p:Product{name:"GroundNuts",production:"50%"}) return p;

create (p:Product{name:"Cotton",production:"50%"}) return p;
```

```
match (c:Country),(s:States),(p:Product) where c.name="India" and s.name="Maharashtra" and
p.name="Wheat" create (c)-[:has_States]->(s)-[:Produces]->(p) return c,s,p;

match (c:Country),(s:States),(p:Product) where c.name="USA" and s.name="California" and
p.name="Oil" create (c)-[:has_States]->(s)-[:Produces]->(p) return c,s,p;

match (c:Country),(s:States),(p:Product) where c.name="India" and s.name="Maharashtra" and
p.name="Oil" create (c)-[:has_States]->(s)-[:Produces]->(p) return c,s,p;

match (c:Country),(s:States),(p:Product) where c.name="India" and s.name="Maharashtra" and
p.name="GroundNuts" create (c)-[:has_States]->(s)-[:Produces]->(p) return c,s,p;

match (c:Country),(s:States),(p:Product) where c.name="India" and s.name="Punjab" and
p.name="Sugar" create (c)-[:has_States]->(s)-[:Produces]->(p) return c,s,p;
```

Queries

1) List the countries that export oil

```
 match (c:Country),(p:Product) where p.name="Oil" and (c)-[:exports]->(p) return c.name;
```

2) List the products produced in "Maharashtra"

```
match (s:States),(p:Product) where s.name="Maharashtra" and (s)-[:Produces]->(p) return p.name;
```

Slip no 14

14.Design an abstract class Employee with computeSal() as an abstract function. Create two subclasses Worker and Manager. Salary of worker should be calculated on hourly basis of work and Salary of Manager should be calculated on monthly basis with additional incentives. Create five objects each of Worker and Manager class, and display their details

```
abstract class Employee
{
def computeSalary():Int;
}
class manager(var mno:Int,var mname:String,var month:Int) extends Employee
{
def computeSalary():Int=
{
var salary=month*30000;
return salary;
}
def display()
{
println("-----------------------------------
--------------------------------");
println("Manager No:"+mno);
println("Manager Name:"+mname);
println("Salary:"+computeSalary());
}
}
class worker(var wno:Int,var wname:String,var hr:Int) extends Employee
{
def computeSalary():Int=
{
var salary=hr*300;
return salary;
}
def display()
{
println("-----------------------------------
-----------------------------------");
println("Worker No:"+wno);
println("Worker Name:"+wname);
println("Salary:"+computeSalary());
}
}
object Slip14
{
def main(args:Array[String])
{
var manager1=new manager(1,"Akshay Borse",4);
var manager2=new manager(2,"Sumit Amritkar",3);
var manager3=new manager(3," Vishnu Khatale",2);
var manager4=new manager(4,"Roshan Adke",5);
var manager5=new manager(5,"Tushar Amrutkar",9);
var worker1=new worker(1,"Ganesh Darade",12);
var worker2=new worker(2,"Viraj Gadekar",16);
var worker3=new worker(3," Abhi Chavhan",10);
var worker4=new worker(4,"Kalpesh Deshmukh",5);
var worker5=new worker(5,"Abhijit Rakibe",4);
println("-----------------------MANAGER------------------
--------------------------");
manager1.display();
manager2.display();
manager3.display();
manager4.display();
manager5.display();
println("-----------------------WORKER-------------------
-------------------------");
worker1.display();
worker2.display();
worker3.display();
worker4.display();
worker5.display();
}
}
```

2 Model the following Furniture Showroom information as a graph model, and answer the queries using Cypher. Consider a Furniture showroom with different types of furnitures like sofas sets, tea tables, cupboards, beds, dining tables, etc. Showroom is divided into different sections, one section for each furniture type, Each section handled by a sales staff. A sales staff can handle one or more sections. Customer may enquire about furniture. An enquiry may result in a purchase by the customer.

```
create(f:Furniture{name:"Sofa Set", color:"white"}) return f;

create(f:Furniture{name:"Tea Table", color:"Black"}) return f;

create(f:Furniture{name:"Cupboards", color:"Brown"}) return f;

create(f:Furniture{name:"beds", color:"white"}) return f;

create(f:Furniture{name:"Dining Table", color:"Black"}) return f;

create(s:Staff{name:"Satish"}) return s;

create(s:Staff{name:"Suraj"}) return s;

create(s:Staff{name:"Omkar"}) return s;

create(s:Staff{name:"Pratik"}) return s;
```

```
match(s:Staff),(f:Furniture) where s.name="Satish" and  f.name="Sofa Set" create (s)-[:Handled]->(f) return s,f;
```

```
match(s:Staff),(f:Furniture) where s.name="Suraj" and  f.name="Tea Table" create (s)-[:Handled]->(f) return s,f;
```

```
match(s:Staff),(f:Furniture) where s.name="Omkar" and  f.name="Cupboards" create (s)-[:Handled]->(f) return s,f;
```

```
match(s:Staff),(f:Furniture) where s.name="Pratik" and  f.name="beds" create (s)-[:Handled]->(f) return s,f;
```

```
create(c:Customer{name:"Priyanshu"}) return c;
```

```
create(c:Customer{name:"Sahil"}) return c;
```

```
match(c:Customer),(f:Furniture) where c.name="Sahil" and  f.name="Sofa Set" create (c)-[:Enquired_about]->(f) return c,f;
```

```
match(c:Customer),(f:Furniture) where c.name="Priyanshu" and  f.name="Dining Table" create (c)-[:Enquired_about]->(f) return c,f;
```

Quries

1) List the types of furniture's available in white colour.

```
match(f:Furniture) where f.color="white" return f.name;
```

2) List the sections handled by Mr. Satish.

```
match(f:Furniture),(s:Staff) where s.name="Satish" and (s)-[:Handled]->(f) return f.name;
```

Slip no 15

15.Write a program to create a list of 1 to 100 numbers. Create second list from first list selecting numbers which are perfect square. Display it.

```
import scala.collection.mutable.ListBuffer
object Slip15
{
def main(args:Array[String])
{
val l1=List.range(1,101);
var l2:ListBuffer[Int]=ListBuffer();
for(i<-l1)
{
for(j<-1 to i)
{
if(i==j*j)
l2+=i;
}
}
println("Perfect Numbers:"+l2);
}
}
```

2) Model the following Clothing Brand information as a graph model, and answer the following queries using Cypher. Consider a Mall for clothing. This mall will include different sections for males, females and kids. Each section contains different types of apparels from different brands. There are many apparels with different designs, of each type. An apparel may be available in one or more standard sizes (S/M/L/XL/XXL)

```
create (s:Section {name:"Male"}) return s;

create (s:Section {name:"Female"}) return s;

create (s:Section {name:"Kids"}) return s;

create (a:Apparel {name:"Kurta"}) return a;

create (a:Apparel {name:"Saree"}) return a;

create (a:Apparel {name:"T-Shirts"}) return a;

create (a:Apparel {name:"Jackets"}) return a;

create (a:Apparel {name:"Frock"}) return a;

create (a:Apparel {name:"Shirt"}) return a;

create (s:Size {name:"S"}) return s;
```

create (s:Size {name:"M"}) return s;

create (s:Size {name:"L"}) return s;

match(s:Section),(a:Apparel) where s.name="Female" and a.name="Kurta" create (s)-[:Has]->(a) return s,a;

match(s:Section),(a:Apparel) where s.name="Female" and a.name="Saree" create (s)-[:Has]->(a) return s,a;

match(s:Section),(a:Apparel) where s.name="Male" and a.name="Jackets" create (s)-[:Has]->(a) return s,a;

match(s:Section),(a:Apparel) where s.name="Male" and a.name="Jackets" create (s)-[:Has]->(a) return s,a;

match(s:Section),(a:Apparel) where s.name="Kids" and a.name="Frock" create (s)-[:Has]->(a) return s,a;

create (ss:SalesStaff {name:"Smita"}) return ss;

create (ss:SalesStaff {name:"Geeta"}) return ss;

create (ss:SalesStaff {name:"Seeta"}) return ss;

create (ss:SalesStaff {name:"Raman"}) return ss;

match(ss:SalesStaff),(s:Section) where s.name="Kids" and ss.name="Smita" create (ss)-[:Work_in]->(s) return ss,s;

match(ss:SalesStaff),(s:Section) where s.name="Female" and ss.name="Geeta" create (ss)-[:Work_in]->(s) return ss,s;

match(ss:SalesStaff),(s:Section) where s.name="Male" and ss.name="Seeta" create (ss)-[:Work_in]->(s) return ss,s;

match(ss:SalesStaff),(s:Section) where s.name="Kids" and ss.name="Raman" create (ss)-[:Work_in]->(s) return ss,s;

Queries

1)List the different apparels type in female section

match(s:Section),(a:Apparel) where s.name="Female" and (s)-[:Has]->(a) return a.name;

2) List the names of sales staff in Kids section.

match(ss:SalesStaff),(s:Section) where s.name="Kids" and (ss)-[:Work_in]->(s) return ss.name;

Slip no 16

16.Write user defined functions to reverse the case of a given string and call the function using MAP.

```
object Slip16
{
def reverse(ch:Char):Char=
{
if(ch.isLower)
ch.toUpper;
else
ch.toLower;
}
def main(args:Array[String])
{
var str=" ";
println("Enter String:");
str=scala.io.StdIn.readLine();
var str2=str.map(reverse);
println(str2);
}
}
```

2 Model the following Hotels information as a graph model, and answer the following queries using Cypher. Consider hotels in Pune. Some hotels provide lodging facility whereas some provide only restaurant facility and some provide both. A person can rate(1-5 stars) a hotel for its facility/facilities. A person can recommend a hotel to his/her friends. A person can provide a review for a hotel after his stay/visit.

```
create (h:Hotel {name:"Marriotte", location:"Camp"}) return h;

create (h:Hotel {name:"Blue Diamond", location:"KP Road"}) return h;

create (h:Hotel {name:"Radison", location:"Kharadi"}) return h;

create (f:Facility {name:"Lodging",rating:"4 star"}) return f;

create (f:Facility {name:"Restaurant ",rating:"4 star"}) return f;

create (f:Facility {name:"Lodging Restaurant ",rating:"5 star"}) return f;

match (h:Hotel),(f:Facility) where h.name="Blue Diamond" and f.name="Lodging Restaurant " create (h)-[:Has]->(f) return h,f;

match (h:Hotel),(f:Facility) where h.name="Radison" and f.name="Restaurant " create (h)-[:Has]->(f) return h,f;

match (h:Hotel),(f:Facility) where h.name="Marriotte" and f.name="Lodging Restaurant " create (h)-[:Has]->(f) return h,f;
```

Queries

1)List the names of hotels in Camp area.

```
match (h:Hotel) where h.location="Camp" return h.name;
```

2) List the names of hotels having both lodging and restaurant facility

```
match (h:Hotel),(f:Facility) where f.name="Lodging Restaurant " and (h)-[:Has]->(f) return h.name;
```

Slip no 17

17.Define a class SavingAccount (accNo, name, balance, minBalance).Define appropriate constructors and operations withdraw(), deposit(),viewBalance(). Create an array of SavingAccount objects and perform operations and display them.

```
class SavingAccount(var acno:Int,var name:String,var balance:Int,var
minbalance:Int)
{
def withdraw()
{
println("Enter Amount:");
var n1=scala.io.StdIn.readInt();
balance=balance-n1;
if(balance<minbalance)
{
println("TRANSACTION FAILED:");
balance=balance+n1;
}
else
println("TRANSACTION SUCCESSFULL");
}
def deposite()
{
println("Enter Amount:");
var n1=scala.io.StdIn.readInt();
balance=balance+n1;
}
def viewbalance()
{
println("Account Number:"+acno);
println("Name:"+name);
println("Balance:"+balance);
println("Minimum Balance:"+minbalance);
}
}
object Slip17
{
def main(args:Array[String])
{
val s1=new Array[SavingAccount](5)
var ch=0;
s1(0)=new SavingAccount(1,"Akshay Borse",20000,10000);
s1(1)=new SavingAccount(2,"Sumit Amritkar",30000,15000);
s1(2)=new SavingAccount(3,"Vishnu Khatale",40000,6000);
s1(3)=new SavingAccount(4,"Ganesh Darade",50000,3000);
s1(4)=new SavingAccount(5,"Tushar Amrutkar",55000,10000);
println("Enter Account Number:");
var ac=scala.io.StdIn.readInt();
for(i<-0 to 4)
{
if(s1(i).acno==ac)
{
println("Account number Exsists");
println("1.Cash Withdraw:");
println("2.Cash Deposite:");
println("3.View Balance:");
println("4.Exit");
while(ch!=5)
{
println("Enter Your
Choice:");
var
ch=scala.io.StdIn.readInt();
ch match
{
case
1=>s1(i).withdraw();
case
2=>s1(i).deposite();
case
3=>s1(i).viewbalance();
case
4=>System.exit(1);
}
}
}
}
println()
}
}
```

2 Model the following Hospitals information as a graph model, and answer the following queries using Cypher. Consider hospitals in and around Pune. Each hospital may have one or more specializations like Pediatric, Gynaec, Orthopedic, etc. A person can recommend/provide review for a hospital. A doctor can be associated with one or more hospitals.

```
create (h:Hospital {name:"Columbia"}) return h;

create (h:Hospital {name:"Rubi"}) return h;

create (h:Hospital {name:"Sahyadri"}) return h;

create(s:Specialization{name:"Pediatric"}) return s;

create(s:Specialization{name:"Orthopedic"}) return s;

create(s:Specialization{name:"Gynaec"}) return s;
```

```
match(h:Hospital),(s:Specialization) where h.name="Columbia" and s.name="Gynaec" create (h)-
[:Specialized_in]->(s) return h,s;
```

```
match(h:Hospital),(s:Specialization) where h.name="Sahyadri" and s.name="Orthopedic" create (h)-
[:Specialized_in]->(s) return h,s;
```

```
match(h:Hospital),(s:Specialization) where h.name="Rubi" and s.name="Pediatric" create (h)-
[:Specialized_in]->(s) return h,s;
```

```
create (p:Person {name:"Vishal"}) return p;
```

```
create (r:Recommend {name:"Service"}) return r;
```

```
match(h:Hospital),(r:Recommend) where h.name="Columbia" and r.name="Service" create (h)-
[:Recommend_as]->(r) return h,r;
```

```
create (d:Doctor {name:"Aarohi"}) return d;
```

```
create (d:Doctor {name:"Rohit"}) return d;
```

```
match (d:Doctor),(h:Hospital) where d.name="Rohit" and h.name="Columbia" create (d)-
[:Associated_with]->(h) return d,h;
```

```
match (d:Doctor),(h:Hospital) where d.name="Rohit" and h.name="Rubi" create (d)-[:Associated_with]-
>(h) return d,h;
```

```
match (d:Doctor),(h:Hospital) where d.name="Aarohi" and h.name="Columbia" create (d)-
[:Associated_with]->(h) return d,h;
```

Queries

1)List the names of hospitals with paediatric specialization.

```
match (h:Hospital),(r:Specialization) where r.name="Pediatric" and (h)-[:Specialized_in]->(r) return r,h;
```

2)List the Names of doctors who are visiting “Ruby Hospital

```
match (h:Hospital),(d:Doctor) where h.name="Rubi" and (d)-[:Associated_with]->(h) return d.name;
```

Slip no 18

18.Write a program to calculate sum of all perfect numbers between 1 and 100. Display perfect numbers also.

```
object Slip18
{
def main(args:Array[String])
{
var sum=0;
var psum=0;
var perfect=" ";
for(i<-1 to 100)
{
for(j<-1 to i-1)
{
if(i%j==0)
{
sum=sum+j;
}
}
if(sum==i)
{
psum=psum+i;//sum of perfect number;
perfect=perfect+" "+i;
}
sum=0;
}
println("perfectNumbers:"+perfect);
println("Sum of Perfect Number:"+psum);
}
}
```

2 Model the following Doctor’s information system as a graph model, and answer the following queries using Cypher. Consider the doctors in and around Pune. Each Doctor is specialized in some stream like Pediatric, Gynaec, Heart Specialist, Cancer Specialist, ENT, etc. A doctor may be a visiting doctor across many hospitals or he may own a clinic. A person can provide a review/can recommend a doctor.

```
create (s:Streams {name:"Pediatric"}) return s;

create (s:Streams {name:"Heart Specialist"}) return s;

create (s:Streams {name:"Cancer"}) return s;

create (s:Streams {name:"Neurosurgery"}) return s;

create (d:Doctor {name:"Aarohi",type:"Own Clinic"}) return d;

create (d:Doctor {name:"Rohit",type:"Visiting Hospital"}) return d;
create (d:Doctor {name:"Sanket",type:"Own Clinic"}) return d;

create (d:Doctor {name:"Krishna",type:"Visiting Hospital"}) return d;

create (p:Person {name:"Vishal"}) return p;

create (p:Person {name:"Swaraj"}) return p;

create (r:Recommend {name:"Service"}) return r;

create (r:Recommend {name:"Recovery"}) return r;

match(d:Doctor),(r:Recommend) where d.name="Sanket" and r.name="Recovery" create (d)-
[:Recommend_as]->(r) return d,r;

match(d:Doctor),(r:Recommend) where d.name="Rohit" and r.name="Service" create (d)-
[:Recommend_as]->(r) return d,r;

match(d:Doctor),(s:Streams) where d.name="Aarohi" and s.name="Neurosurgery" create (d)-[:Is_Of]-
>(s) return d,s;

match(d:Doctor),(s:Streams) where d.name="Sanket" and s.name="Cancer" create (d)-[:Is_Of]->(s)
return d,s;

match(d:Doctor),(s:Streams) where d.name="Rohit" and s.name="Heart Specialist" create (d)-[:Is_Of]-
>(s) return d,s;

match(d:Doctor),(s:Streams) where d.name="Krishna" and s.name="Neurosurgery" create (d)-[:Is_Of]-
>(s) return d,s;
```

Queries

1) List the Neurosurgery doctors.

```
match(d:Doctor),(s:Streams) where s.name="Neurosurgery" and (d)-[:Is_Of]->(s) return d.name;
```

2)List the doctors who own a clinic.

```
match(d:Doctor),(s:Streams) where d.type="Own Clinic" return d.name;
```

Slip no 19

```
import scala.util._object Prg19

{

def main(args:Array[String])

{

var l1:List[Int]=List();

var l2:List[Int]=List();

var n1=0;

for(i<- 1 to 10)

{

l1::=Random.nextInt(10);

}

println("List1:"+l1.sorted);

for(j<-l1)

{

n1=3*j*j+4*j+6;

l2::=n1;

}

println("list2:"+l2.sorted)

}

}
```

2 Model the following Automobile information system as a graph model, and answer the following queries using Cypher. Consider an Automobile industry manufacturing different types of vehicles like Heavy Vehicles, Light Vehicles, etc. A customer can buy one or more types of vehicle. A person can recommend or rate a vehicle type.

```
create(i:Industry{name:"KTM"})return i;

create(v:VehicleType{name:"Heavy"})return v;

create(v:VehicleType{name:"Light"})return v;

match(i:Industry),(v:VehicleType) where i.name="KTM" and v.name="Heavy" create(i)-[:Has]->(v) return
i,v;

match(i:Industry),(v:VehicleType) where i.name="KTM" and v.name="Light" create(i)-[:Has]->(v) return
i,v;

create(vv:VehicleChar{name:"Duke",color:"Orange",wheels:"two wheeler"})return vv;

create(vv:VehicleChar{name:"RC",color:"White",wheels:"two wheeler"})return vv;

match(v:VehicleType),(vv:VehicleChar) where v.name="Heavy" and vv.name="Duke" create (v)-
[:has_char]->(vv) return vv,v;

match(v:VehicleType),(vv:VehicleChar) where v.name="Light" and vv.name="Activa" create (v)-
[:has_char]->(vv) return vv,v;

create(c:Customer {name:"Shivani"}) return c;

create(c:Customer {name:"Pooja"}) return c;

match(c:Customer),(v:VehicleType) where c.name="Shivani" and v.name="Light" create (c)-[:Bought]-
>(v) return c,v;

match(c:Customer),(v:VehicleType) where c.name="Shivani" and v.name="Heavy" create (c)-[:Bought]-
>(v) return c,v;

match(c:Customer),(v:VehicleType) where c.name="Pooja" and v.name="Light" create (c)-[:Bought]->(v)
return c,v;
```

Queries

1)List the characteristics of heavy vehicle types.

```
match(v:VehicleType),(vv:VehicleChar) where v.name="Heavy" and (v)-[:has_char]->(vv) return vv.name,vv.color,vv.wheels;
```

2) List the name of customers who bought a heavy vehicle.

```
match(c:Customer),(v:VehicleType) where v.name="Heavy" and (c)-[:Bought]->(v) return c.name;
```

Slip no 20

20.Create a list of 10 random numbers. Create another list from members of first list using function 3n2+4n+6. Display second list in ascending order.

```
import scala.util._
object Slip20
{
def main(args:Array[String])
{
var l1:List[Int]=List();
var l2:List[Int]=List();
var n1=0;
for(i<- 1 to 10)
{
l1::=Random.nextInt(10);
}
println("List1:"+l1.sorted);
for(j<-l1)
{
n1=3*j*j+4*j+6;
l2::=n1;
}
println("list2:"+l2.sorted)
}
}
```

2 Model the following University information system as a graph model, and answer the following queries using Cypher. University has various departments like Mathematics, Geology, Chemistry, etc. Each department conducts various courses and a course may be conducted by multiple departments.

```
create(u:University{name:"Pune University"}) return u;

create(d:Department{name:"Chemistry"}) return d;

create(d:Department{name:"Geology"}) return d;

create(c:Course{name:"BSc"}) return c;

create(c:Course{name:"MSc"}) return c;

match (u:University),(d:Department) where d.name="Chemistry" and u.name="Pune University" create
(u)-[:has]->(d) return u,d;

match (u:University),(d:Department) where d.name="Geology" and u.name="Pune University" create
(u)-[:has]->(d) return u,d;
```

```
match (c:Course),(d:Department) where c.name="MSc" and d.name="Chemistry" create (d)-[:conduct]->(c) return d,c;
```

```
match (c:Course),(d:Department) where c.name="BSc" and d.name="Geology" create (d)-[:conduct]->(c) return d,c;
```

```
match (c:Course),(d:Department) where c.name="BSc" and d.name="Chemistry" create (d)-[:conduct]->(c) return d,c;
```

Queries

1) List the names of the courses provided by Chemistry Department.

```
match (c:Course),(d:Department) where d.name="Chemistry" and (d)-[:conduct]->(c) return c.name;
```

2)List the details of all the departments in the university.

```
match (u:University),(d:Department) where u.name="Pune University" create (u)-[:has]->(d) return d.name;
```