1.Design database using simple data structure.

Algorithm:

Step1: Start (create a class named student)

Step2: the method of the class,getstudentdetails()gets input from the user print result() calculates result and prints it.

Step3: we will also add extra marks(9) to a student

Step4: then print the result

Step5: Stop

Input:

```
class Student:
def getStudentDetails(self,i):
self.rollno=input("Enter Roll Number: ")
Self.name = input("Enter Name: ")
self,physics =int(input("Enter Physics Marks: "))
self.chemistry = int(input("Enter Chemistry Marks: "))
self.maths = int(input("Enter Math Marks: "))
def display(self,i):
print("RollNo: ", self.rollno)
print("Name: ",self.name)
print("Score1: ", self.physics)
print("Score2: ", self.chemistry)
print("Score3: ",self.maths)
print("n")
def printResult(self):
self.percentage = (int)( (self.physics+ self.chemistry + self.maths)/300 * 100);
print(self.rollno,self.name, self.percentage)
n=intinput("Enter how many students records"))
for i in range(n):
```

```
S1-Student()
51.getStudentDetails(i)
print("\nList of Students\n")
for i in range(n):
$1.displayli)
print("Result: ")
S1.printResult()
S1.physics +9
print("result after adding grace marks...")
S1.printResult()
```

Output:

```
Enter how many students records
3
Enter Roll Number: 1
Enter Name: a
Enter Physics Marks : 45
Enter Chemistry Marks : 67
Enter Math Marks : 78
Enter Roll Number: 2
Enter Name : b
Enter Physics Marks : 56
Enter Chemistry Marks : 78
Enter Math Marks : 89
Enter Roll Number: 3
Enter Name:c
Enter Physics Marks : 67
Enter Chemistry Marks : 78
Enter Math Marks : 89
```

List of Students

RollNo: 3

Name : c

Score1 : 67

Score2 : 78

Score3: 89

RollNo: 3

Name: c

Scorel: 67

Score2 : 78

Score3: 89

RollNo: 3

Name: c

Score1: 67

Result:

3 c 78

result after adding grace marks...

3 c 81

Working:

In this problem we need to get input from users for the rollno,name and marks of three subjects.then we need to calculate the percentage and print result, and reprint it after giving extra marks.

2.Implementation of student database using Abstract data types (ADT)

Algorithm:

Step 1 :- Start

Step2 :- create a class named Student

Step3. - using init fuctions and self argument give the detail name, rollno, m1, m2 marks. create the functions and view them to give values.

Step4 :- print every junction name and give the values.

Step5: perform accept, delete Update, exit operations.

Step6:Stop

Input:

```
class student:
def init_(self, name, rollno, m1, m2):
selt.name=name
self.rollno=rollno
self.m1=m1
self.m2=m2
def accept(self, name, rollno, marks1, marks2): ob=student(name, rolino, marks1, marks2))
Is.append(ob)
def display(self, ob):
print("Name", ob.name)
print("Rollno ", ob.rollno)
print("Marks1:", ob.m1)
print("Marks2:", ob.m2)
print("\n")
def search(self, rn):
for i in range(ls._len_0):
if(ls[i].rolino==rn):
return i
def delete(self, rn):
i=obj.search (rn)
del Is[i]
def update(self, rn, no):
i=obj.search(rn)
roll=no
Is[i].rollno=roll;
```

```
Is=1
obj=student(''0,0,0)
print("nOperations used, ")
print("n1.accept student details\n2.display student details \n"
"3.search details of a student\n4.delete details of student"
"\n5.update student details\n6.exit")
print("1->ACCEPT")
obj.accept("A", 1, 100, 100)
obj.accept("B", 2, 90, 90)
obj.accept("C", 3, 80, 80)
print("press any key to continue.."
input()
1.acc.
2.dis
3.S
print("2->DISPLAY")
print("press any key to continue...")
input()
print("\n")
print("\nlist of student\n")
for i in range(ls._len_0):
obj.display(ls[i])
print("3->SEARCH")
print("press any key to continue... ")
input()
print("\n student found,")
s=obj.search(2)
obj.display(ls[s])
print("4->DELETE")
```

```
print("press any key to continue... ")
input()
obj.delete(2)
print(ls_len_())
print("list after deletion")
for i in range(ls.len_():
obj.display(ls[i])
print("5->UPDATE")
print("press any key to continue... ")
input()
obj.update(3,2)
print(ls_en_(0)
print("list after updation")
for i in range(ls. len_():
obj.display(s[il)
print("6->EXIT")
print("press any key to continue...")
input()
print("Thank you !")
```

Output:

Operations used,

[8:34 PM, 6/12/2023] Smartestbadboy: 1.accept student details

2.display student details

3.search details of a student

4.delete details of student

5.update student details

6.exit

1-->ACCEPT

press any key to continue...

2-->DISPLAY

press any key to continue...

list of student

Name : A

Rollno : 1

Marks1 : 100

Marks2 : 100

Name : B

Rollno :2

Marks1:90

Marks2 :90

Name : C

Rollno :3

Marks1:80

Marks2: 80

3-->SEARCH

press any key to continue...

press any key to continue...

student found,

Name : B

Rollno :2

Marks1 : 90

Marks2: 90

4-->DELETE

press any key to continue...

[8:36 PM, 6/12/2023] Smartestbadboy: 2

list after deletion

Name : A

Rollno : 1

Marks1 : 100

Marks2 : 100

Name : C

Rollno :3

Marks1:80

Marks2 : 80

5-->UPDATE

press any key to continue...

2

list after updation

Name : A

Rollno : 1

Marks1 : 100

Marks2 : 100

Name : C

Rollno : 2

Marks1 : 80

Marks2 : 80

6-->EXIT

press any key to continue...

Thank you !

Working:

Operations used,

1.Accept Student details

2.Display Student Details

3.Search Details of a Student

4.Delete Details of Student

5.Update Student Details

6.Exit

It gives the output of all used functions with students marks and the grace marks added.

3.write a program to perform push and pop Operations using list has a stack.

Algorithm:

Step 1 :- Start

Step 2 :- Store the element to push into armay.

/Step 3 :- check if top = = (marsize - 1) then atale

else goto step 4.

Skep4:- Increment top as top=top +1

Step5 :- Add element to the position stack[top]=num

Step 6 :- Stop

Input:

```
stack=[]
def push():
item=input("enter the item to be pushed")
stack.append(item)
print(stack)
def pop():
if not stack:
print("stack is empty")
else:
e=stack.pop()
print("the element poped is =",e)
true=1
while true:
```

```
print("1.push 2.pop 3.exit")
ch=int(input("enter the choice"))
if(ch==1):
push()
elif(ch==2): pop() elif(ch==3): break
else:
print("enter the correct choice")
```

Output:

```
1.push 2.pop 3.exit
enter the choicel
enter the item to be pushed20
['20']
1.push 2.pop 3.exit
enter the choice1
enter the item to be pushed60
['20', '60']
1.push 2.pop 3.exit
enter the choice1
enter the item to be pushed45
['20', '60', '45']
1.push 2.pop 3.exit
enter the choice2
the element poped is = 45
1.push 2.pop 3.exit
enter the choice3
```

Working:

In the above code, we have defined an empty list. We inserted the elements one by one using the append() method. That is similar to the push() method. We also removed the elements using the pop() method. The pop() method returns the last element of the list.

4.write a program to perform all operations on queue using list has a queue.

Algorithm:

Step 1:- Start

Step 2 :- Implementing queue using list : q = [] q. append (10)

Step 3 :- implement queue using list funitions) q = []

def enqueue (); if len (q) = -size + check wheather

queue in full or not..

Step 4: implement queue using queue module. from queue

Import queue of q=Queue (max size 4).....

Steps :- from collection Implement operations like pop()

display () and exit ()

Step 6 – Stop

Input:

```
queue=[]
def push():
print("enter the element to push")
item=input()
queue.append(item)
print(item, "placed in queue")
def pop():
if not queue:
print("queue is empty")
else:
item=queue.pop()
print("the element poped is",item)
```

```
def display():
print(queue)
true=1
while true:
print("the operations on queue are")
print("1.push 2.pop 3.display ")
ch=int(input("enter the choice"))
if(ch==1):
push()
elif(ch==2):
pop()
elif(ch==3):
display()
else:
print("enter correct choice")
```

Output:

```
the operations on queue are
1.push 2.pop 3.display
enter the choice1
enter the element to push
20
20 placed in queue
the operations on queue are
1.push 2.pop 3.display
enter the choice1
enter the element to push
45
45 placed in queue
```

the operations on queue are

1.push 2.pop 3.display

enter the choice3

['20', '45']

the operations on queue are

1.push 2.pop 3.display

enter the choice2

the element poped is 45

the operations on queue are

1.push 2.pop 3.display

Working:

In the above, we have imported the queue module that is a LIFOqueue. It works the same as the stack but this module includes some additional functions mentioned above. We defined a stack with the maxsize that means it can hold maximum five values in it.

The initial array size is zero; we pushed three elements in the stack using the put() method. Now, again we checked whether a stack is empty and size of the stack. We have three elements in the stack. We popped the element using the get() method. It removes the last added element first. After removing the entire elements, we get an empty queue.