Abstract Factory:

```
import random
class Course_At_GFG:
    def __init__(self, courses_factory=None):
        self.course_factory = courses_factory
    def show_course(self):
        course = self.course_factory()
        print(f'We have a course named {course}')
        print(f'its price is {course.Fee()}')
class DSA:
    def Fee(self):
        return 11000
    def __str__(self):
        return "DSA"
class STL:
    def Fee(self):
        return 8000
    def __str__(self):
        return "STL"
class SDE:
    def Fee(self):
        return 15000
    def __str__(self):
        return 'SDE'
def random_course():
    return random.choice([SDE, STL, DSA])()
if __name__ == "__main__":
    course = Course_At_GFG(random_course)
    for i in range(5):
        course.show_course()
```

adapter:

```
class MotorCycle:
    def __init__(self):
        self.name = "MotorCycle"

    def TwoWheeler(self):
        return "TwoWheeler"

class Truck:
    def __init__(self):
        self.name = "Truck"

    def EightWheeler(self):
        return "EightWheeler"

class Car:
    def __init__(self):
        self.name = "Car"

    def FourWheeler(self):
        return "FourWheeler"

class Adapter:
    def __init__(self, obj, **adapted_methods):
        self.obj = obj
        self.__dict__.update(adapted_methods)

    def __getattr__(self, attr):
        return getattr(self.obj, attr)

    def original_dict(self):
        return self.obj.__dict__

if __name__ == "__main__":
    objects = []

    motorCycle = MotorCycle()
    objects.append(Adapter(motorCycle, wheels=motorCycle.TwoWheeler))

    truck = Truck()
    objects.append(Adapter(truck, wheels=truck.EightWheeler))

    car = Car()
    objects.append(Adapter(car, wheels=car.FourWheeler))

    for obj in objects:
        print("A {0} is a {1} vehicle".format(obj.name, obj.wheels()))
```

Bridge:

```
class Cuboid:

    class ProducingAPI1:
        def produceCuboid(self, length, breadth, height):
            print(f'API1 is producing Cuboid with length = {length}, '
                f' Breadth = {breadth} and Height = {height}')

    class ProducingAPI2:
        def produceCuboid(self, length, breadth, height):
            print(f'API2 is producing Cuboid with length = {length}, '
                f' Breadth = {breadth} and Height = {height}')

    def __init__(self, length, breadth, height):
        self._length = length
        self._breadth = breadth
        self._height = height

    def produceWithAPI1(self):
        objectAPIone = self.ProducingAPI1()
        objectAPIone.produceCuboid(self._length, self._breadth, self._height)

    def producewithAPI2(self):
        objectAPItwo = self.ProducingAPI2()
        objectAPItwo.produceCuboid(self._length, self._breadth, self._height)

    def expand(self, times):
        self._length *= times
        self._breadth *= times
        self._height *= times

cuboid1 = Cuboid(1, 2, 3)
cuboid1.produceWithAPI1()

cuboid2 = Cuboid(19, 20, 21)
cuboid2.producewithAPI2()
```

Command:

```
from abc import ABC, abstractmethod

class Command(ABC):
    def __init__(self, receiver):
        self.receiver = receiver

    def process(self):
        pass

class CommandImplementation(Command):
    def __init__(self, receiver):
        self.receiver = receiver

    def process(self):
        self.receiver.perform_action()

class Receiver:
    def perform_action(self):
        print('Action performed in receiver.')

class Invoker:
    def command(self, cmd):
        self.cmd = cmd

    def execute(self):
        self.cmd.process()

if __name__ == "__main__":
    receiver = Receiver()
    cmd = CommandImplementation(receiver)
    invoker = Invoker()
    invoker.command(cmd)
    invoker.execute()
```

Composite:

```
class LeafElement:
    def __init__(self, *args):
        self.position = args[0]

    def showDetails(self):
        print("\t", end="")
        print(self.position)

class CompositeElement:
    def __init__(self, *args):
        self.position = args[0]
        self.children = []

    def add(self, child):
        self.children.append(child)

    def remove(self, child):
        self.children.remove(child)

    def showDetails(self):
        print(self.position)
        for child in self.children:
            print("\t", end="")
            child.showDetails()

if __name__ == "__main__":
    topLevelMenu = CompositeElement("GeneralManager")
    subMenuItem1 = CompositeElement("Manager1")
    subMenuItem2 = CompositeElement("Manager2")
    subMenuItem11 = LeafElement("Developer11")
    subMenuItem12 = LeafElement("Developer12")
    subMenuItem21 = LeafElement("Developer21")
    subMenuItem22 = LeafElement("Developer22")

    subMenuItem1.add(subMenuItem11)
    subMenuItem1.add(subMenuItem12)
    subMenuItem2.add(subMenuItem21)
    subMenuItem2.add(subMenuItem22)

    topLevelMenu.add(subMenuItem1)
    topLevelMenu.add(subMenuItem2)
    topLevelMenu.showDetails()
```

Façade :

```
class Washing:
    def wash(self):
        print("Washing...")

class Rinsing:
    def rinse(self):
        print("Rinsing...")

class Spinning:
    def spin(self):
        print("Spinning...")

class WashingMachine:
    def __init__(self):
        self.washing = Washing()
        self.rinsing = Rinsing()
        self.spinning = Spinning()

    def startWashing(self):
        self.washing.wash()
        self.rinsing.rinse()
        self.spinning.spin()

if __name__ == "__main__":
    washingMachine = WashingMachine()
    washingMachine.startWashing()
```

Observer :

```
class Subject:
    def __init__(self):
        self._observers = []

    def notify(self, modifier=None):
        for observer in self._observers:
            if modifier != observer:
                observer.update(self)

    def attach(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        try:
            self._observers.remove(observer)
        except ValueError:
            pass

class Data(Subject):
    def __init__(self, name=''):
        Subject.__init__(self)
        self.name = name
        self._data = 0

    @property
    def data(self):
        return self._data

    @data.setter
    def data(self, value):
        self._data = value
        self.notify()

class HexViewer:
    def update(self, subject):
        print('HexViewer: Subject {} has data 0x{:x}'.format(subject.name, subject.data))

class OctalViewer:
    def update(self, subject):
        print('OctalViewer: Subject {} has data {}'.format(subject.name, oct(subject.data)))

class DecimalViewer:
    def update(self, subject):
        print('DecimalViewer: Subject {} has data {}'.format(subject.name, subject.data))

if __name__ == "__main__":
```

```
obj1 = Data('Data 1')
obj2 = Data('Data 2')

view1 = DecimalViewer()
view2 = HexViewer()
view3 = OctalViewer()
obj1.attach(view1)
obj1.attach(view2)
obj1.attach(view3)

obj2.attach(view1)
obj2.attach(view2)
obj2.attach(view3)

obj1.data = 10
obj2.data = 15
```

Proxy:

```
class College:
    def studyingInCollege(self):
        print("Studying In College....")

class CollegeProxy:
    def __init__(self):
        self.feeBalance = 1000
        self.college = None

    def studyingInCollege(self):
        print("Proxy in action. Checking to see if the balance of student is clear or not...")
        if self.feeBalance <= 500:
            self.college = College()
            self.college.studyingInCollege()
        else:
            print("Your fee balance is greater than 500, first pay the fee")

if __name__ == "__main__":
    collegeProxy = CollegeProxy()
    collegeProxy.studyingInCollege()
    collegeProxy.feeBalance = 100
    collegeProxy.studyingInCollege()
```

Strategy:

```
class Item:
    def __init__(self, price, discount_strategy=None):
        self.price = price
        self.discount_strategy = discount_strategy

    def price_after_discount(self):
        if self.discount_strategy:
            discount = self.discount_strategy(self)
        else:
            discount = 0
        return self.price - discount

    def __repr__(self):
        statement = "Price: {}, price after discount: {}"
        return statement.format(self.price, self.price_after_discount())

def on_sale_discount(order):
    return order.price * 0.25

def twenty_percent_discount(order):
    return order.price * 0.20

if __name__ == "__main__":
    print(Item(20000))
    print(Item(20000, discount_strategy=twenty_percent_discount))
    print(Item(20000, discount_strategy=on_sale_discount))
```

Bayes:

```
pAF = int(input())
print("The probability that it is Friday and that a student is absent :", pAF)

pF = int(input())
print("The probability that it is Friday : ", pF)

pResult = (pAF / pF)

print("The probability that a student is absent given that today is Friday :", pResult * 100, "%")
```

Sql connector:

```
#pip install mysql-connector-python
"""
CREATE DATABASE STUDENTDATA;

USE STUDENTDATA;

CREATE TABLE STUDENT(
ROLLNO INT,
NAME VARCHAR(100),
AGE INT
);

INSERT INTO STUDENT(ROLLNO, NAME, AGE) VALUES
(1, 'AAKASH', 23),
(2, 'ALICE', 22),
(3, 'BOB', 21);

SELECT * FROM STUDENT;
"""
import mysql.connector

# Establish connection to the MySQL database
conn = mysql.connector.connect(
   host="localhost",      # or your database server IP/domain
   user="root",   # your MySQL username
   password="root", # your MySQL password
   database="STUDENTDATA"  # name of the database you want to connect to
)

# Check if the connection is successful
if conn.is_connected():
   print("Connected to MySQL database")
   cursor = conn.cursor()

   # Step 3: Execute SQL query
   cursor.execute("SELECT * FROM student")

   # Step 4: Fetch all rows
   rows = cursor.fetchall()

   #C:\Users\AAKASH BONAGIRI\OneDrive\Desktop\ML\lab\sql.py

   # Step 5: Print the results
   print("Student Table Data: (ROLLNO, NAME, AGE)")
   for row in rows:
      print(row)
# Close the connection
conn.close()
```