#question1

```
def get_non_zero_input(prompt):
    """Gets user input, ensuring it's not zero."""
    while True:
        try:
            value = int(input(prompt))
            if value == 0:
                raise ValueError("Input cannot be zero.")
            return value
        except ValueError:
            print("Invalid input. Please enter a non-zero integer.")

def main():
    num1 = get_non_zero_input("Enter first number: ")
    num2 = get_non_zero_input("Enter second number: ")

    # Arithmetic operators
    print("Arithmetic Operators:")
    print(f"  Sum: {num1} + {num2} = {num1 + num2}")
    print(f"  Difference: {num1} - {num2} = {num1 - num2}")
    print(f"  Product: {num1} * {num2} = {num1 * num2}")
    print(f"  Division: {num1} / {num2} = {num1 / num2:.2f}")
    print(f"  Floor Division: {num1} // {num2} = {num1 // num2}")
    print(f"  Modulo: {num1} % {num2} = {num1 % num2}")
    print(f"  Power: {num1} ** {num2} = {num1 ** num2}")

    # Logical operators
    print("\nLogical Operators:")
    print(f"  AND: {True} and {True} = {True and True}")
    print(f"  AND: {True} and {False} = {True and False}")
    print(f"  OR: {True} or {False} = {True or False}")
    print(f"  OR: {False} or {False} = {False or False}")
    print(f"  NOT: not {True} = {not True}")

    # Bitwise operators
    print("\nBitwise Operators:")
    print(f"  AND: {num1} & {num2} = {num1 & num2}")
    print(f"  OR: {num1} | {num2} = {num1 | num2}")
    print(f"  NOT: ~{num1} = {~num1}")
    print(f"  XOR: {num1} ^ {num2} = {num1 ^ num2}")

# Call the main function directly
main()
```

#question2

```
pip install mysql-connector-python                    #execute separate cell
```

```
#execute separate cell
```

```
import mysql.connector as sql
db = sql.connect(host="localhost",user="root",password="root",database="db")
c = db.cursor()
while(True):
    print("1. Create table")
    print("2. Insert rows")
    print("3. Update rows")
    print("4. Delete rows")
    print("5. Display")
    print("6. Exit")

    opt = int(input("Enter your choice: "))
    if opt == 1:
        c.execute("Create table if not exists student(snum int,sname varchar(10),m1 int,m2 int,total int)")
        print("Table created \n")
    elif opt == 2:
        v1 = int(input("Enter snum: "))
        v2 = input("Enter student name: ")
        v3 = int(input("Enter m1: "))
        v4 = int(input("Enter m2: "))
        v5 = v3+v4
        v = [v1,v2,v3,v4,v5]
        sql = "Insert into student(snum,sname,m1,m2,total) values (%s,%s,%s,%s,%s)"
        c.execute(sql,v)
        db.commit()
        print(c.rowcount," rows inserted\n")
    elif opt == 3:
        v1 = int(input("Enter snum whose mark must be updated: "))
        v3 = int(input("Enter m1: "))
        v4 = int(input("Enter m2: "))
        v5 = v3+v4
        v = [v3,v4,v5,v1]
        sql = "Update student set m1 = %s , m2 = %s, total = %s where snum = %s"
        c.execute(sql,v)
        db.commit()
        print(c.rowcount," rows updated\n")
    elif opt == 4:
        v=input("Enter the person name to be deleted:")
        val=(v,)
        sql = "DELETE FROM student WHERE snum = %s"
        c.execute(sql, val)
        db.commit()
        print(c.rowcount,"record(s) deleted \n")
    elif opt == 5:
        c.execute("Select * from student")
        res = c.fetchall()
        for i in res:
            print(i)
        print(c.rowcount," rows displayed\n")
    else:
        db.close()
        print("Exit the program")
        break
```

#question3

```
def reverse():
    num=int(input("Enter a number:"))
    print("The given number is ",num)
    sum=0
    print("Reversed number is ",end="")
    while(num>0):
        r=num%10
        print(r,end="")
        sum=sum+r
        num//=10 #num=num//10
    print("\nSum of its digits is",sum)
reverse()
```

#question4

```
def prime():
    try:
        n = int(input("Enter the n value: "))
        if n < 0:
            raise ValueError("n is not a number")

        print("Prime Numbers from 1 to {} are".format(n))

        for i in range(1, n + 1):
            flag = 0
            for j in range(2, i):
                if i % j == 0:
                    flag += 1
            if flag == 0 and i > 1:
                print(i, end=" ")
    except ValueError as ve:
        print("Error:", ve)
prime()
```

#question5

```
def perfect(start, end):
    perfect_numbers = []
    for num in range(start, end + 1):
        sum_factors = 0
        for i in range(1, num):
            if num % i == 0:
                sum_factors += i
        if sum_factors == num:
            perfect_numbers.append(num)
    if len(perfect_numbers) == 0:
        print("There are no perfect numbers in the given range.")
    else:
```

```
        print("Perfect numbers in the range", start, "to", end, "are:", perfect_numbers)



print("FINDING PERFECT NUMBER SERIES")
start = int(input("Enter the starting number: "))
end = int(input("Enter the ending number: "))
perfect(start, end)

#question6
def quadratic(a, b, c):
    if a != 0:
        d = (b ** 2) - (4 * a * c)
        if d == 0:
            res = -b / (2 * a)
            return res
        elif d > 0:
            res1 = (-b + (d) ** 0.5) / (2 * a)
            res2 = (-b - (d) ** 0.5) / (2 * a)
            return res1, res2
        else:
            return "It is a complex number"
    else:
        return "The given equation is not quadratic"

print("\t\tROOTS OF QUADRATIC EQUATION")
no1 = int(input("Enter the coefficient of x^2: "))
no2 = int(input("Enter the coefficient of x: "))
no3 = int(input("Enter the coefficient of constant: "))
answer = quadratic(no1, no2, no3)
print("The result is:", answer)

#question7
# Opening file in append mode to retain the data
f = open("Employee.dat", "a")

# Loop to input employee details and write them to the file
for i in range(2):
    empnum = input("Enter the employee number: ")
    empname = input("Enter the employee name: ")
    row = empnum + "," + empname + "\n"
    f.write(row)

f.close()

# Reading contents from "Employee.dat" and copying to "Emp.dat" in reverse order
with open("Employee.dat", "r") as file:
    lines = file.readlines()
```

```
with open("Emp.dat", "w") as file:
    file.writelines(reversed(lines))

print("Employee details written to 'Employee.dat' and copied to 'Emp.dat' in reverse order.")
```

#question8

```
class Sample:
    def __init__(self, var):
        self.var = var

    # Arithmetic Operator Overloading
    def __sub__(self, other):
        return self.var - other.var

    def __mul__(self, other):
        return self.var * other.var

    # Relational Operator Overloading
    def __lt__(self, other):
        return self.var < other.var

    def __gt__(self, other):
        return self.var > other.var

    def __eq__(self, other):
        return self.var == other.var

    # Bitwise Operator Overloading
    def __and__(self, other):
        return self.var & other.var

    def __or__(self, other):
        return self.var | other.var


# Example usage:

obj1 = Sample(5)
obj2 = Sample(3)

# Arithmetic Operator Overloading
print("Subtraction:", obj1 - obj2)  # Output: 2
print("Multiplication:", obj1 * obj2)  # Output: 15
# Relational Operator Overloading
print("Less Than:", obj1 < obj2)  # Output: False
```

```
print("Greater Than:", obj1 > obj2)  # Output: True
print("Equal To:", obj1 == obj2)  # Output: False
# Bitwise Operator Overloading
print("Bitwise AND:", obj1 & obj2)  # Output: 1
print("Bitwise OR:", obj1 | obj2)  # Output: 7
```

#question9

```
#create
d = {'a':45,'b':78,'c':34,'d':80,'e':67}
print("Dictionary: ",d)

#keys and values
print("Keys: ")
for i in d.keys():
    print(i)
print("Values: ")
for i in d.values():
    print(i)

#max and min
maximum = max(d,key = d.get)
minimum = min(d,key = d.get)
print("\nStudents scored maximum mark: ",maximum,", marks: ",d[maximum])
print("Students scored minimum mark: ",minimum,", marks: ",d[minimum])

#sort
s = sorted(d.items(),key =lambda x:x[1])
for i,j in s:
    print(i,"-",j)

#modify
d['a'] = 54
print(d)

#delete
dlt = input("Enter the student name: ")
if dlt in d:
    del d[dlt]
    print(dlt," is deleted")
else:
    print(dlt," is not found")

print(d)

print("Number of key-value pairs: ",len(d))
```

#question10

```
import numpy as np
from scipy import linalg
# We are trying to solve a linear algebra system which can be given as
# x + y +z = 2
# 6x -4y + 5z = 31
# 5x + 2y +2z = 13
# Creating input array
a = np.array([[1, 1, 1], [6, -4, 5], [5, 2, 2]])
b = np.array([[2], [31], [13]])
# Solve the linear algebra
result = linalg.solve(a, b)
# Print results
print("x={} y={} z={}".format(int(result[0]),int(result[1]),int(result[2])))

#Finding Determinant of the given matrix
#  4 -3 0
#  2 -1 -2
#  1 5 7

from scipy import linalg
import numpy as np
#Declaring the numpy array
A = np.array([[4 ,-3, 0],[2 ,-1, -2],[1 ,5 ,7]])
#Passing the values to the det function
x = linalg.det(A)
#printing the result
print("The Determinant of the given matrix is ",x)
```

#question11

```
#creation of tuple
T = ('red' , 'blue' , 'green' , 'yellow' , 'pink')
print("Tuple:" , T)

#1.justifying tuple is immutable
T[0]
#T[0] = 'orange'
print("Tuple:" , t)

#2. print individual elements of tuple
for i in T:
    print (i)

#3. Create two tuples with T1 as colors starting with “b” and others in T2
T1 = ('beige', 'blue' , 'black', 'Beige')
T2 = ('teal','pink','gray','purple')

#4. Check if “orange” is in T or not.
```

```
if "orange" in T:
    print("Yes, Orange is in Tuplpe T")
else:
    print("No, Orange is not in Tuple T")

#5.Can tuple allow duplicates?
print(T1)
```

#question12

```
def check_palindrome(s):
    sr = s[::-1]
    if s == sr:
        print("Palindrome")
    else:
        print("Not a palindrome")

def compare_strings(s):
    s1 = input("Enter another string: ")
    if s == s1:
        print("Equal")
    else:
        print("Not equal")

s = input("Enter a string: ")
check_palindrome(s)
print("Length: ", len(s))
compare_strings(s)

my_str = "Consider function f(n) the time complexity of an algorithm and g(n) is the most significant
term. If f(n) <= C g(n) for all n >= 1, C > 0, then we can represent f(n) as O(g(n))"
print(my_str.count("the"))
upper, lower, digits, spaces, specials = 0, 0, 0, 0, 0
for c in my_str:
    if c.isupper():
        upper += 1
    elif c.islower():
        lower += 1
    elif c.isdigit():
        digits += 1
    elif c.isspace():
        spaces += 1
    else:
        specials += 1
print("Upper:", upper, "Lower:", lower, "Digits:", digits, "Spaces:", spaces, "Specials:", specials)
print(my_str.swapcase())
```

#question13

```
# Function to calculate the amount
def calculate_amount(unit_price, quantity):
    return unit_price * quantity
```

```
# Open the file in write mode
with open("Inventory.txt", "w") as file:
    # Loop to input details for five products
    for i in range(2):
        # Input product details
        pnum = input("Enter product number: ")
        pname = input("Enter product name: ")
        unit_price = float(input("Enter unit price: "))
        quantity = int(input("Enter quantity: "))

        # Calculate the amount
        amount = calculate_amount(unit_price, quantity)

        # Write product details and amount to the file
        file.write(f"{pnum},{pname},{unit_price},{quantity},{amount}\n")

# Open the file again to read and display its contents
with open("Inventory.txt", "r") as file:
    print("Contents of Inventory.txt:")
    for line in file:
        print(line.strip())
```

#question14

main file

```
import module1 as m

# Example list of numbers
numbers = [23, 45, 67, 12, 90]

# Finding maximum
max_value = m.find_maximum(numbers)
print("Maximum value:", max_value)

# Finding minimum
min_value = m.find_minimum(numbers)
print("Minimum value:", min_value)

# Finding sum
sum_value = m.find_sum(numbers)
print("Sum:", sum_value)

# Finding average
average_value = m.find_average(numbers)
print("Average:", average_value)
```

module1.py

```
def find_maximum(numbers):
    """Find the maximum value in a list of numbers."""
    if not numbers:
        return None
    return max(numbers)
```

```
def find_minimum(numbers):
    """Find the minimum value in a list of numbers."""
    if not numbers:
        return None
    return min(numbers)

def find_sum(numbers):
    """Calculate the sum of all numbers in a list."""
    return sum(numbers)

def find_average(numbers):
    """Calculate the average of numbers in a list."""
    if not numbers:
        return None
    return sum(numbers) / len(numbers)
```

#question15

```
l = []
for i in range(1, 11):
    print("No",i,":",end="")
    n = int(input())
    l.append(n)
print("Original list:", l)

# Sorting in ascending order
asc = sorted(l)
print("Ascending order:", asc)

# Sorting in descending order
dsc = sorted(l, reverse=True)
print("Descending order:", dsc)

def div3(l):
    l1 = []
    l2 = []
    for i in l:
        if i % 3 == 0:
            l1.append(i)
        else:
            l2.append(i)
    return l1, l2

divisible_by_3, not_divisible_by_3 = div3(l)
print("Divisible by 3:", divisible_by_3)
print("Not divisible by 3:", not_divisible_by_3)

max_num = max(divisible_by_3)
min_num = min(not_divisible_by_3)

divisible_by_3.remove(max_num)
```

```
not_divisible_by_3.remove(min_num)

print("After removing the biggest number from divisible_by_3:", divisible_by_3)
print("After removing the smallest number from not_divisible_by_3:", not_divisible_by_3)
```

#question16

```
import numpy as np

a = np.array([[1,2,3],[4,5,6],[6,7,8]])
b = np.array([[6,5,4],[2,4,6],[7,9,2]])
add = a + b
mul = a @ b

# Iterate over the elements of 'add'
print("Element-wise addition:")
for x in np.nditer(add):
    print(x, end=" ")

# Iterate over the elements of 'mul'
print("\nMatrix multiplication:")
for x in np.nditer(mul):
    print(x, end=" ")
```

#question17

```
class car:
    total_cars = 0
    def __init__(self,brand,model):
        car.total_cars+=1
        self.brand = brand
        self.model = model
    def display_info(self):
        print("Brand: ",self.brand)
        print("Model: ",self.model)
    def __del__(self):
        car.total_cars-=1
        print("{} and {} is destroyed".format(self.brand,self.model))

car11 = car("Tesla",'Model S')
car21 = car('Ford','Fusion')
car11.display_info()
print("Total: ",car.total_cars)
del car21
print("Total: ",car.total_cars)
```

#question18

```
# Create sets
Deepak = {"Python", "Java", "C", "C++"}
Uma = {"PHP", "SQL", "ASP.NET", "C"}

# Display the sets
print("Deepak's languages:", Deepak)
print("Uma's languages:", Uma)

# Find common languages
common_languages = Deepak.intersection(Uma)
print("Common languages:", common_languages)

# List all languages known by both
all_languages = Deepak.union(Uma)
print("All languages known by both:", all_languages)

# List languages known by Deepak but not by Uma
deepak_only = Deepak.difference(Uma)
print("Languages known by Deepak only:", deepak_only)

# List languages known by Uma but not by Deepak
uma_only = Uma.difference(Deepak)
print("Languages known by Uma only:", uma_only)

# Add "Go" to Deepak
Deepak.add("Go")
print("Deepak's languages after adding 'Go':", Deepak)

# Remove "SQL" from Uma
Uma.remove("SQL")
print("Uma's languages after removing 'SQL':", Uma)
```

#question19

```
import numpy as np
a = np.arange(5,10)
print(a)
print("Size: ",a.size)
print("Dimension: ",a.ndim)
print("Max: ",np.max(a))
print("Min: ",np.min(a))
print("Sum: ",np.sum(a))
print("Mean: ",np.mean(a))
print("Median: ",np.median(a))
print("SD: ",np.std(a))
print("Var: ",np.var(a))
```

#question20

```
#Hybrid Inheritance

#class 1
class Base1:
    def get1(self):
        self.snum = int(input("Enter the Student Roll No:"))
        self.sname = input("Enter the Student Name:")

    def put1(self):
        print("Student Rollno: ", self.snum)
        print("Student Name: ", self.sname)

#child class of class 1
class Base2(Base1):
    def get2(self):
        #inheriting the fuctions of parent class
        Base1.get1(self)
        Base1.put1(self)

        self.mark1 = int(input("Enter the Mark1:"))
        self.mark2 = int(input("Enter the Mark2:"))

    def put2(self):
        print("Mark1: ", self.mark1)
        print("Mark2: ", self.mark2)

#independent class
#class 2
class Base3:
    def get3(self):
        self.score = int(input("Enter the score: "))
    def put3(self):
        print("Score: ", self.score)


#child class of class 1 and class 2
class Child(Base2, Base3):
    def put4(self):
        #inheriting the fuctions of parent classes
        Base2.get2(self)
        Base2.put2(self)
        Base3.get3(self)
        Base3.put3(self)

        self.total = self.mark1 + self.mark2 + self.score
        print("The total score of the student: ",self.total)
```

```
s = Child()
s.put4()
```

#question21

```
#multiple Inheritance

#parent class1
class Base1:
    def get1(self):
        self.num1 = int(input("Enter Number 1: "))
    def put1(self):
        print("Number 1: ", self.num1)

#parent class2
class Base2:
    def get2(self):
        self.num2 = int(input("Enter Number 2: "))
    def put2(self):
        print("Number 2: ", self.num2)

#child class
class Child(Base1, Base2):
    def put3(self):
         #inheriting the fuctions of parent classes
        Base1.get1(self)
        Base2.get2(self)
        Base1.put1(self)
        Base2.put2(self)

        print("Addition: " , self.num1 + self.num2)
        print("Subtraction: " , self.num1 - self.num2)
        print("Multiplication: " , self.num1 * self.num2)
        print("Division: " , self.num1 / self.num2)
        print("Floor Division: " , self.num1 // self.num2)
        print("Modulus: " , self.num1 % self.num2)
        print("Exponentation: " , self.num1 ** self.num2)

        if self.num1 > self.num2:
            print(self.num1 , "is the greatest number")
        else:
            print(self.num2 , "is the greatest number")

c = Child()
c.put3()
```

#question22

```
#multilevel inheritance

#parent class
class Empbase:
    def get1(self):
        self.enum = int(input("Enter Employee ID: "))
        self.ename = input("Enter Employee Name: ")
        self.basic = int(input("Enter the Basic Pay:"))
    def put1(self):
        print("Employee ID: ",self.enum)
        print("Employee Name: ",self.ename)
        print("Basic Pay: ",self.basic)

#child class
class Empchild1(Empbase):
    def get2(self):
        #inheriting the fuctions of parent classes
        Empbase.get1(self)
        Empbase.put1(self)

        self.allow = int(input("Enter Allowance: "))
        self.ded = int(input("Enter Deduction: "))
        self.gross = self.basic + self.allow
        self.net = self.gross - self.ded

    def put2(self):
        print("Allowance: ",self.allow)
        print("Deduction: ",self.ded)

class Empchild2(Empchild1):
    def get3(self):
        #inheriting the fuctions of parent classes
        Empchild1.get2(self)
        Empchild1.put2(self)

        self.gross = self.basic + self.allow
        self.net = self.gross - self.ded
    def put3(self):
        print("Gross: ",self.gross)
        print("Net: ",self.net)

#main
obj = Empchild2()
obj.get3()
obj.put3()
```

#question23

```
#single inheritance

#parent class
class Empbase:
    def get(self):
        self.enum = int(input("Enter Employee ID: "))
        self.ename = input("Enter Employee Name: ")
        self.basic = int(input("Enter the Basic Pay:"))
    def put(self):
        print("Employee ID: ",self.enum)
        print("Employee Name: ",self.ename)
        print("Basic Pay: ",self.basic)

#child class
class Empchild(Empbase):
    def get(self):
        #inheriting the fuctions of parent classes
        Empbase.get(self)
        Empbase.put(self)

        self.allow = int(input("Enter Allowance: "))
        self.ded = int(input("Enter Deduction: "))
        self.gross = self.basic + self.allow
        self.net = self.gross - self.ded

    def put(self):
        print("Gross: ",self.gross)
        print("Net: ",self.net)

#main
obj = Empchild()
obj.get()
obj.put()
```

#question24

```
#Hierarchical inheritance

class Base:
    def get(self):
        self.num1 = int(input("Enter Number 1: "))
        self.num2 = int(input("Enter Number 2: "))
    def put(self):
        print("Number 1: ", self.num1)
        print("Number 2: ", self.num2)

class Child1(Base):
```

```
    def put1(self):
        #inheriting the fuctions of parent classes
        Base.get(self)
        Base.put(self)

        print("Arithmetic Operations")
        print("Addition: " , self.num1 + self.num2)
        print("Subtraction: " , self.num1 - self.num2)
        print("Multiplication: " , self.num1 * self.num2)
        print("Division: " , self.num1 / self.num2)
        print("Floor Division: " , self.num1 // self.num2)
        print("Modulus: " , self.num1 % self.num2)
        print("Exponentation: " , self.num1 ** self.num2)

class Child2(Base):
    def put2(self):
        #inheriting the fuctions of parent classes
        Base.get(self)
        Base.put(self)

        print("Logical Operations")
        print("Logical and: ",a and b)
        print("Logical or: ",a or b)
        print("Logical not: ", not a)
        print("Logical not: ", not b)

c1 = Child1()
c2 = Child2()
c1.put1()
c2.put2()
```

#question25

```
#creation
import pandas as pd
data =
{'Courses':["Spark","PySpark","Hadoop","Python","Pandas",None,"Spark","Python"],'Fee'
:[22000,25000,23000,24000,np.nan,25000,25000,22000],
'Duration':[30,50,55,40,60,35,45,50],'Discount':[1000,2300,1000,1200,2500,1300,1400,1600]
}
# displaying the DataFrame
df=pd.DataFrame(data)
df
#1
df.shape
#2
df.dtypes
#3
```

```
df.iloc[[1, 3, 5], [0, 2]]
#4
df.loc[(df['Discount'] > 1000) & (df['Discount'] <2000),['Courses','Discount']]
#5
df['Tutors']=['William', 'Henry', 'Michael', 'John', 'Messi', 'Ramana','Kumar','Vasu']
df
#6
df=df.rename(columns={'Fee': 'Fees'})
df
#7
df.isnull().sum()
#8
df[(df.Tutors.str.startswith('P'))]
#9
df.loc[(df['Duration'] > 40) ,['Courses','Duration']]
#10
df.groupby('Courses')[['Fees', 'Discount']].mean()
```

#question26

```
import pandas as pd
d = {"country": ["Brazil", "Russia", "India", "China", "South Africa"], "capital": ["Brasilia",
"Moscow", "New Delhi", "Beijing", "Pretoria"], "area": [8.516, 17.10, 3.286, 9.597, 1.221],
"population": [200.4, 143.5, 1252, 1357, 52.98]}
d
d1 = pd.DataFrame(d)
#1
d1.sort_values(by="population",ascending=False)
#2
import matplotlib.pyplot as plt
d1["area"].plot.pie(autopct='%1.1f%%', labels=d1['country'])
plt.title('area vs country')
plt.ylabel('')
plt.show()
#3
d1.plot.bar(x='country', y='population',color='red')
plt.title('country vs population')
plt.xlabel('country')
plt.ylabel('population')
plt.show()
#4
d1[['country','population']]
#5
d1[d1['country'] == 'Russia']['capital'].values[0]
d1[d1['country'] == 'Russia']['capital']
#6
d1[d1['capital'].str.endswith('a')]
#7
d1.isnull()
```

```
#8
d1.sort_values(by='area').iloc[0]['country']
d1.sort_values(by='population').iloc[0]['country']
#9
d1.query('area > 7')
#10
d1.columns
```

#question27

```
#creation
import pandas as pd
data =
{'Employee':['Sahay','George','Priya','Manila','Raina','Manila','Priya'],'Sales':[125600,235600,
213400,189000,456000,172000,201400],'Quarter':[1,1,1,1,1,2,2],'State':['Delhi','Tamil
Nadu','Kerala','Haryana','West Bengal','Haryana','Kerala']}
# displaying the DataFrame
df=pd.DataFrame(data)
df
#1
df.loc[(df['Quarter'] == 1),['Quarter','State']]
#2
df.loc[(df['Quarter'] == 2),['Quarter','Employee']]
#3
df[['Employee','State']]
#4
df.loc[(df['Sales'] > 200000),['Employee','Sales']]
#5
df.groupby('State')['Sales'].sum()
#6
df.groupby('Quarter')['Sales'].agg(['mean', 'median', 'max', 'min'])
#7
df[df['Sales'] > df[df['State'] == 'Kerala']['Sales'].mean()]
#8
df['Employee'].unique()
#9
df[df.State.str.contains('e')]['State'].unique()
#10
import matplotlib.pyplot as plt
df.plot.bar(x='Employee', y='Sales',color='blue')
plt.title('Employee vs Sales')
plt.xlabel('Employee')
plt.ylabel('Sales')
plt.show()
```

#question28

```
import pandas as pd
```

```
data = {"Name": ["Asha", "Harsh", "Sourav", "Hritik", "Shivansh", "Akash", "Soumya",
"Karthik"],
   "Dept": ["Administration", "Marketing", "Technical", "Technical", "Administration",
"Marketing", "Technical", "Administration"],
   "Type": ["Fulltime", "Intern", "Intern", "Parttime", "Parttime", "Fulltime", "Intern", "Intern"],
   "Salary": [120000, 50000, 70000, 67800, 55000, 57900, 64300, 110000],
   "Years": [10, 2, 3, 4, 7, 3, 2, 8]}
df=pd.DataFrame(data)
df

#1
df.groupby('Type')['Name'].apply(list)
#2
df.loc[(df['Dept'] == 'Technical') & (df['Type'] == 'Parttime')]
#3
df.groupby('Dept').agg({'Salary': ['mean', 'sum']})
#4
df.loc[(df['Years'] > 2)]
#5
df.info()
#6
df[df['Type'] == 'Intern'].nlargest(1, 'Salary')
#7
plt.bar(df['Name'], df['Years'], color='skyblue')
plt.xlabel('Employee')
plt.ylabel('Experience (Years)')
plt.title('Employee Experience')
plt.show()
#8
df.nsmallest(1, 'Salary')['Name']
#9
df.nsmallest(1, 'Salary')
#10
dept=df['Dept'].unique()
print(len(dept))
print(dept)
```

#question29

```
import pandas as pd
data = {"age": [10,22,13,21,12,11,17],
   "section": ["A","B","C","B","B","A","A"],
   "city": ["Gurgaon","Delhi","Mumbai","Delhi","Mumbai","Delhi","Mumbai"],
   "gender": ["M","F","F","M","M","M","F"],
   "favourite_color": ["red",np.nan,"yellow",np.nan,"black","green","red"]}
df=pd.DataFrame(data)
df
```

```
#1
df.groupby(['city', 'favourite_color']).size()
#2
df[df['age'] < 20][['gender', 'favourite_color']]
#3
df[df['city'].str.endswith('i')]['city']
#4
df.isnull().sum()
#5
df.fillna(value="orange")
#6
df['city'].unique()
#7
gender_counts = df['gender'].value_counts()

print("Number of males:", gender_counts['M'])
print("Number of females:", gender_counts['F'])
#8
df.groupby('city')['age'].mean()
#9
df.groupby('section')['age'].sum()
#10
df['city'].value_counts()
```

#question30

```
import pandas as pd
data = {"Name": ["John","Jane","Emily","Lisa","Matt"],
   "Note": [92,94,87,82,90],
   "Profession": ["Electrical engineer","Mechanical engineer","Data
Scientist","Accountant","Athlete"],
   "date_of_birth": ["1998-11-01","2002-08-14","1996-01-12","2002-10-24","2004-04-05"],
   "group": ["A","B","B","A","C"]}
df=pd.DataFrame(data)
df
#1
largest_rows = df.nlargest(2, 'Note')
print("First two largest rows based on 'Note' column:")
print(largest_rows)

smallest_rows = df.nsmallest(2, 'Note')
print("\nFirst two smallest rows based on 'Note' column:")
print(smallest_rows)
#2
df.iloc[[0,1],[0,1]]
#3
df.loc[(df['Note']>90)]
#4
f[df.Profession.str.contains('engineer')][['Name','Profession']]
```

```
#5
df[df['Name'].str.startswith('J')]
#6
df[(df['Profession'] == 'Data Scientist') | (df['Note'] > 90)]
#7
df.iloc[[2,3,4],[2]]
#8
df[df['group'].isin(['A', 'C'])]['Name']
#9
df[df['Profession'] == 'Athlete']['Name']
#10
df['date_of_birth'] = pd.to_datetime(df['date_of_birth'])

# Filter the DataFrame for persons born after 2000
filtered_df = df[df['date_of_birth'].dt.year > 2000]
print(filtered_df)
```