1. A.

```
      class ZeroError(Exception):
    def __init__(self, num, message = "There integer must be greater than 0"):
        self.num = num
        self.message = message
        super().__init__(self.message)
    def __str__(self):
        return f"{self.message} : Provided value is {self.num}"
def arth_log_bitwise(a,b):
    print("Arithmetic operations :")
    print(f"{a} + {b}= {a+b}")
    print(f"{a} - {b}= {a-b}")
    print(f"{a} * {b}= {a*b}")
    print(f"{a} / {b}= {a/b} \n")
    print("Logical operations :")
    print(f"{a}>{b} and {a} < {b} = {(a>b) and (a<b)}")
    print(f"{a}<{b} or {a} > {b} = {(a<b) or (a>b)}")
    print(f"not {a} < {b} = {not(a<b)} \n")
    print("Bitwise operations")
    print(f"{a} & {b} = {a & b}")
    print(f"{a} | {b} = {a | b}")
    print(f"{a} ^ {b} = {a ^ b}")
    print(f"{a} << {b} = {a << b}")
    print(f"{a} >> {b} = {a >> b}")
    print(f"~{b} = {~b}")
try:
    a = int(input("Enter a integer value :"))
    b = int(input("Enter a integer value :"))
    if a == 0:
        raise ZeroError(a)
    elif  b == 0:
        raise ZeroError(b)
    else:
        arth_log_bitwise(a, b)
except ZeroError as e :
    print(e.message)
    print(e)
    print(f"But provided :{e.num}")
```

1.B.

Ans:

Open mysql → CREATE DATABASE college;

→ USE college;

Jupiter → Cell 1 → !pip install mysql-connector-python

Cell 2:

```
import mysql.connector as sql
mycon = sql.connect(host="localhost", user="root", password="", database="college")
if mycon.is_connected():
    while(True):
        print("\nWelcome to database Program \nChoose a Choice to do operations \nEnter 1 to create table \nEnter 2 to insert the rows \nEnter 3 Update the rows \nEnter 4 to Delete the rows \nEnter 5 to Display the rows \n6 To Exit")

        ch = int(input("Enter a choice :"))
        mycur = mycon.cursor()

        if(ch == 1):
            qry = "CREATE TABLE IF NOT EXISTS student(snum integer(2) , sname varchar(20) , m1 integer(3) , m2 integer(3) , total integer(3) )"
            mycur.execute(qry)
            print("\nTable successfully created ")
        elif(ch == 2):
            num = int(input("Enter the rollNo :"))
            name = input("Enter your name :")
            m1 = int(input("Enter your mark for m1 :"))
            m2 = int(input("Enter your mark for m1 :"))
            tot = m1 + m2
            #qry = f"INSERT into student values({num} , '{name}',{m1},{m2},{tot})"
            qry = "INSERT into student values(%s , %s , %s , %s , %s)"
            mycur.execute(qry,[num , name , m1 , m2, tot])
            chk = mycur.rowcount
            mycon.commit()
            if chk > 0:
                print("\nSuccessfully inserted")
        elif(ch == 3):
            num = int(input("Enter the roll no to update :"))
            name = input("Enter the updated name :")
            m1 = int(input("Enter the updated m1 mark :"))
            m2 = int(input("Enter the updated m2 mark :"))
            tot = m1 + m2
            qry = f"UPDATE student set sname = %s , m1 = %s , m2 = %s , total = %s where snum = {num}"
            mycur.execute(qry, [name, m1, m2, tot])
            mycon.commit()
            print("\nUpdated Successfully")
        elif(ch == 4):
            num = int(input("Enter the roll no to delete :"))
            qry = "Delete from student where snum = %s"
            mycur.execute(qry,(num ,))
            mycon.commit()
            print("\nDeleted successfully")
        elif(ch == 5):
            mycur.execute("select * from student")
            print("The rows are:")
            rows = mycur.fetchall()
            for i in rows:
                print(i)
        elif(ch == 6):
               break
else:
    print("Not connected")
```

<u>**Mysql prompt:**</u>

```
USE college;
SHOW TABLES;
SELECT * FROM student;
DESCRIBE student;
DROP TABLE student;
USE college;
DROP DATABASE IF EXISTS college;
```

2.A.

```
def check_perfect_number(num):
    lst =[]
    for j in range(1,num +1):
        summ=0
        for i in range(1,j):
            if (j % i==0):
                summ += i
        if (j == summ):
            lst.append(j)

    print(f"The perfect number is {lst}")
n=eval(input("Enter a range number: "))
check_perfect_number(n)
```

2.B.

Answer:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
    "Name": ["Asha", "Harsh", "Sourav", "Hritik", "Shivansh", "Akash", "Soumya", "Kartik"],
    "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],
    "Exp": [10, 2, 3, 4, 7, 3, 2, 8]
}
df = pd.DataFrame(data)
print("Number of employees in each department:\n", df["Dept"].value_counts())
parttime = df[(df["Dept"] == "Marketing") & (df["Type"] == "Parttime")]
print("\nPart-time employees in Marketing department:\n", parttime)
print("\nAverage and Total salary of each department :\n" , df.groupby("Dept")["Salary"].agg(["mean",
"sum"]))
print("\n Employee with experience greater than 2 \n" , df[df.Exp > 2] )
print("\nDataset Info:")
print(df.info())
plt.figure(figsize=(10, 5))
plt.bar(df["Name"], df["Exp"], color='green')
plt.xlabel("Employee Name")
plt.ylabel("Experience (Years)")
plt.title("Employee Experience")
plt.xticks(rotation=90)
plt.show()
print("\nUnique Department Names: \n",  df["Dept"].unique())
```

3.A.

```
def divisible_checker(lst):
    l1 = []
    l2 =[]
    for i in lst:
        if i % 3 == 0:
            l1.append(i)
        else:
            l2.append(i)
    return l1 , l2
print("List Operations :")
mylist = [42,23,542,34,4,21]
print("Type of Mylist :",type(mylist))
# for i in range(10):
#     mylist.append(int(input("Enter the integer to add in the list :")))
```

```
mylist.sort()
print("sort in ascending order :",mylist)
mylist.sort(reverse=True)
print("sort in descending order :",mylist)
l1 , l2 = divisible_checker(mylist)
print(f"The List divisible by 3 : {l1}\n The List not divisible by 3 {l2}")
mx = max(l1)
if mx in l1 :
  l1.remove(mx)
mi = min(l2)
if mi in l2:
  l2.remove(mi)
print(f"The List L1 has : {l1} and the removed value is {mx}\n The List L2 has :{l2} and the removed value is {mi}")
```

3.B.

```
class Sample:
  def __init__(self, var):
    self.var = var
  def __sub__(self, other):
    return self.var - other.var
  def __mul__(self, other):
    return self.var * other.var
  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
  def __and__(self, other):
    return self.var & other.var
  def __or__(self, other):
    return self.var | other.var
  # Display method for easy output
```

```
    def __str__(self):
        return str(self.var)
a = Sample(10)
b = Sample(5)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
# Logical operations
print("a < b:", a < b)
print("a > b:", a > b)
print("a == b:", a == b)
# Bitwise operations
print("Bitwise AND:", a & b)
print("Bitwise OR:", a | b)
```

4.A.

```
def check(s):
    pal = s[::-1]
    if(s == pal):
        print(f"The given {s} is palindrome")
        return pal
    else:
        print(f"The given {s} is not palindrome")
        return pal
def count_char(txt):
    lower = upper = space = special = number = 0
    for ch in txt:
        if ch.isupper():
            upper += 1
        elif ch.islower():
            lower +=1
        elif ch.isspace():
            space +=1
        elif ch.isdigit():
            number +=1
        else:
            special += 1

    print(f"The text contains \nUpper :{upper}\nLower :{lower}\nSpace :{space}\nDigits :{number}\nSpecial :{special}\n ")

s = input("Enter the string :")
print(f"Length of the {s} is {len(s)}")
another = check(s)
print("the given two str is equal :" , s== another)
s1 = input("Enter a another string :")
print("Counting how many 'the' in the text")
words = s1.split()
print(words.count("the"))
count_char(s1)
print("Toggle the case :" , s1.swapcase())
```

output:

```
Enter the string :hello
Enter a another string :This is the end. The number is 42!
```

4.B.

```
class base1:
    def get1(self):
        self.snum = int(input("Enter the number :"))
        self.sname = input("Enter the name :")

    def put1(self):
        print(f"snum = {self.snum} \n sname = {self.sname}")

class base2(base1):
    def get2(self):
        self.mark1 =int(input("Enter the mark 1:"))
        self.mark2 = int(input("Enter the mark 2 :"))
    def put2(self):
        print(f"mark1 = {self.mark1} \n mark2 = {self.mark2}")
class base3:
    def get3(self):
        self.score =int(input("Enter the score :"))
    def put3(self):
        print(f"score = {self.score} ")
class child(base2, base3):
    def put4(self):
        self.get1()
        base2.get2(self)
        base3.get3(self)
        self.total = self.mark1 + self.mark2
        self.put1()
        self.put2()
        base3.put3(self)
        print(f"The total is {self.total}")
s1 = child()
s1.put4()
```

5.A.

```
def num_rev():
    num = int(input("Enter the number :"))
    s = 0
    rev =""
    while(num >0):
        r = num % 10
        rev += str(r)
        s += r
        num = num // 10


    print("Sum is ",s , "Rev is ",rev)
num_rev()
```

output:

```
1234
```

5.B.

```
import pandas as pd

import numpy as np

data = {

    'Courses': ["Spark", "PySpark", "Hadoop", "Python", "Pandas", np.nan, "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]

}

df = pd.DataFrame(data)

print(df.loc[[0, 2, 4], ['Courses', 'Duration']])

print()

print(df[(df['Discount'] >= 1000) & (df['Discount'] <= 2000)][['Courses', 'Discount']])

df['Tutors'] = ['William', 'Henry', 'Michael', 'John', 'Messi', 'Ramana', 'Kumar', 'Vasu']

print(df)

df.rename(columns={'Fee': 'Fees'}, inplace=True)

print(df)

print(df.isnull().sum(), "\n")

print(df[df['Courses'].str.startswith('P', na=False)])

print(df[df['Duration'] > 40])

print(df.groupby('Courses')[['Discount', 'Fees']].mean())
```

6.A.

```
def quadratic_equ(a,b,c):

    d=(b**2)-(4*a*c)

    #print(d)

    if(d>0):

        root1=(-b + d**0.5)/(2*a)

        root2=(-b - d**0.5)/(2*a)

        return(f"The Roots For {a},{b},{c} are :\n root 1 is {root1} \n root 2 is {root2}")
    elif(d<0):
        root1= -b/(2*a)
        return(f"The Roots For {a},{b},{c} are :\n root 1 is {root1} \n")
    else:
        return(f"Invalid inputs are given {a},{b},{c}")


a=eval(input("Enter a value for coefficent of square of x : "))
b=eval(input("Enter a value for coefficient of x : "))
c=eval(input("Enter a value for constant : "))
print(quadratic_equ(a, b, c))
```

output:

```
1
2
3
```

6.A.

```
import cmath  # Import cmath to handle complex numbers
def quadratic_equ(a, b, c):
    if a == 0:
        return f"Invalid input: {a} cannot be zero in a quadratic equation."
    d = (b ** 2) - (4 * a * c)  # Compute discriminant


    if d > 0:
        # Two distinct real roots
        root1 = (-b + d ** 0.5) / (2 * a)
        root2 = (-b - d ** 0.5) / (2 * a)
        return f"The Roots For {a}, {b}, {c} are:\nRoot 1: {root1}\nRoot 2: {root2}"
    elif d == 0:
        # One real root (double root)
        root = -b / (2 * a)
        return f"The Roots For {a}, {b}, {c} are:\nDouble Root: {root}"
    else:
        # Complex roots (when d < 0)
    root1 = (-b + cmath.sqrt(d)) / (2 * a)
    root2 = (-b - cmath.sqrt(d)) / (2 * a)
    return f"The Roots For {a}, {b}, {c} are:\nRoot 1: {root1}\nRoot 2: {root2}"
# Take user input
a = float(input("Enter the coefficient of x² (a): "))
b = float(input("Enter the coefficient of x (b): "))
c = float(input("Enter the constant term (c): "))
# Print the result
print(quadratic_equ(a, b, c))
```

output:

```
1
2
3
```

6.B.

```
import pandas as pd
import numpy as np
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)
grouped_colors = df.groupby('city')['favourite_color'].apply(list)
print("Grouped Colors According to City:\n", grouped_colors)
filtered_df = df[df['age'] < 20][['gender', 'favourite_color']]
print("\nGender and Favorite Color of Persons with Age < 20:\n", filtered_df)
df['favourite_color'] = df['favourite_color'].fillna('orange')
print("\nDataFrame after filling NaN in favorite_color:\n", df)
unique_cities = df['city'].unique()
print("\nUnique City Names:\n", unique_cities)
```

```
gender_count = df['gender'].value_counts()
print("\nCount of Males and Females:\n", gender_count)
avg_age_per_city = df.groupby('city')['age'].mean()
print("\nAverage Age Group of Each City:\n", avg_age_per_city)
total_age_per_section = df.groupby('section')['age'].sum()
print("\nTotal Age Per Section:\n", total_age_per_section)
people_per_city = df['city'].value_counts()
print("\nNumber of Persons in Each City:\n", people_per_city)
```

7.A.

```
marks = {'sakthi' : 89 ,'vel':90 , 'chiva' : 67 ,'srini':70,'dhanush' : 56}
print(f"keys : {marks.keys()}")
print(f"values : {marks.values()}")
print(f"Minimarks scored :{min(marks, key=marks.get)}")
print(f"Maximarks scored :{max(marks , key=marks.get)}")
items = sorted(marks.items() , key=lambda x: x[1])
for key , values in items:
    print(key , values)
marks['dhanush'] = 99
print(f"The change dict is {marks}")
del marks['sakthi']
print(f"The change dict is {marks}")
print("\nTotal Key-Value Pairs in Dictionary:", len(marks))
```

7.B.

```
from scipy import linalg
import numpy as np
d = np.array([[1,1,1],[6,-4,5],[5,2,2]])
v = np.array([[2],[31],[13]])
a = linalg.solve(d, v)
print("The result of solve :")
for i in a.flat:
    print(round(i), end=' ')
print()
ch = d.dot(a) - v
for i in ch.flat:
    print(round(i), end=' ')


a = np.array([[4,-3,0],[2,-1,-2],[1,5,7]])
res = linalg.det(a)
print(f"\nDeterminant of \n{a}\n is :{res}")
```

8.A.

```
T = ("blue", "red", "black", "green", "brown")
try:
    T[1] = "yellow"
except TypeError as e:
    print("Tuples are immutable! Error:", e)
print("Individual elements of tuple T:")
for color in T:
    print(color)
l1 =[]
l2 = []
for i in T:
    if i.startswith('b') :
        l1.append(i)
    else:
        l2.append(i)
T1 = tuple(l1)
T2 = tuple(l2)
print("\nTuple T1 (colors starting with 'b'):", T1)
print("Tuple T2 (other colors):", T2)
print("Is 'orange' in T?", "orange" in T)
chk_duplicate = ("blue", "red", "blue", "green", "red")
print("Tuple with duplicates:", chk_duplicate)
```

8.B.

```
class base1:
    def get1(self):
        self.num1 = int(input("Enter the num1 value :"))
    def put1(self):
        print(f"The value of num1 is {self.num1}")
class base2:
    def get2(self):
        self.num2 = int(input("Enter the num2 value :"))
    def put2(self):
        print(f"The value of num1 is {self.num2}")
class child(base1, base2):
    def arthmetic(self):
        print(f"Addition of {self.num1} AND {self.num2} is {self.num1 + self.num2}")
        print(f"Subtraction of {self.num1} AND {self.num2} is {self.num1 - self.num2}")
        print(f"Multiplication of {self.num1} AND {self.num2} is {self.num1 * self.num2}")
        print(f"Division of {self.num1} AND {self.num2} is {self.num1 / self.num2}")
    def find_lar(self):
        self.res = self.num1 if(self.num1 > self.num2) else self.num2
        print(f"Largest of {self.num1} AND {self.num2} is {self.res}")
    def put3(self):
        self.get1()
        self.get2()
        self.put1()
        self.put2()
        print("Arithmetic operations :")
        print('--------------------------------------------')
        self.arthmetic()
        print("Largest Number ")
        print('--------------------------------------------')
        self.find_lar()
obj = child()
obj.put3()
```

output:

```
10

5
```

9.A.

```
class empbase:
    def get1(self):
        self.Enum = int(input("Enter the Enum value :"))
        self.ename = input("Enter the name :")
        self.basic = eval(input("Enter the basic pay :"))
    def put1(self):
        print(f"The value of Enum is {self.Enum}")
        print(f"The value of Name is {self.ename}")
        print(f"The value of Basic pay is {self.basic}")
class empchild1(empbase):
    def get2(self):
        self.ded = eval(input("Enter the Deduction Amount :"))
        self.allowance = eval(input("Enter the allowance Amount :"))
    def put2(self):
        print(f"The value of Deduction is {self.ded}")
        print(f"The value of Allowance is {self.allowance}")
class empchild2(empchild1):
   def get3(self):
       self.gross = self.basic + self.allowance
       self.net = self.gross - self.ded
   def put3(self):
       print(f"The value of gross salary is {self.gross}")
       print(f"The value of net is {self.net}")
   def display(self):
        self.get1()
        self.get2()
        self.get3()
        print("\nDetails")
        print('--------------------------------')
        self.put1()
        self.put2()
        self.put3()
        print('--------------------------------')

obj = empchild2()
obj.display()
```

Output:

```
Enter the Enum value : 101
Enter the name : John
Enter the basic pay : 20000
Enter the Deduction Amount : 2000
Enter the allowance Amount : 5000
```

9.B.

```
import numpy as np
a = np.arange(1,6)
print("The Array:",a,"\nAnd its type :", type(a))
print("Size of array:",a.size)
print("Dimension of array:",a.ndim)
print("Sum of elements:", np.sum(a))
print("Mean of elements:", np.mean(a))
print("Minimum value:", np.min(a))
print("Maximum value:", np.max(a))
```

10.A.

Cell - 1

```
def find_max(numbers):
    return max(numbers)
def find_min(numbers):
    return min(numbers)
def find_sum(numbers):
    return sum(numbers)
def find_average(numbers):
    return sum(numbers) / len(numbers) if numbers else 0
```

Cell-2

```
# main logic
n = int(input("Enter how many numbers: "))
numbers = []
for i in range(n):
    num = float(input(f"Enter number {i+1}: "))
    numbers.append(num)
# Call the functions
maximum = find_max(numbers)
minimum = find_min(numbers)
total = find_sum(numbers)
average = find_average(numbers)
# Display output
print("\n--- Results ---")
print(f"Maximum: {maximum}")
print(f"Minimum: {minimum}")
print(f"Sum: {total}")
print(f"Average: {average}")
```

10.B.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  3 17:06:05 2025
@author: admin
"""
import pandas as pd
import matplotlib.pyplot as plt
# Creating the dataframe
data = {
    "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]
}
df = pd.DataFrame(data)
df_sorted = df.sort_values(by="population", ascending=False)
print("\nCountries sorted by population:\n", df_sorted)


plt.figure(figsize=(7, 7))
explode = [0, 0, 0.1, 0, 0]
plt.pie(df["area"], labels=df["country"], autopct='%1.1f%%', startangle=140, explode=explode)
plt.title("Area Occupied by Each Country")
plt.show()


plt.figure(figsize=(7, 5))
plt.bar(df["country"], df["population"], color=['blue', 'green', 'red', 'purple', 'orange'])
plt.xlabel("Country")
plt.ylabel("Population (millions)")
plt.title("Population of Each Country")
plt.show()


#df_slice = df[["country", "population"]]
df_slice = df.loc[:,["country", "population"]]
print("\nCountry and Population:\n", df_slice)


#capital_russia = df[df["country"] == "Russia"]["capital"].values[0]
capital_russia =df.loc[df["country"] == "Russia", "capital"].iloc[0]
print("\nCapital of Russia:", capital_russia)


#capitals_with_a = df[df["capital"].str.endswith('a')]["capital"].tolist()
capitals_with_a = df.loc[df['capital'].str.endswith('a'), 'capital'].tolist()
print("\nCapitals ending with 'a':", capitals_with_a)
```

```
#smallest_country = df.loc[df["area"].idxmin(), "country"]
smallest_country =df.loc[df["area"].sort_values().index[0], "country"]
print("\nThe smallest country by area is:", smallest_country)

large_countries = df.loc[df["area"] > 7, "country"].tolist()
print("\nCountries with area above 7:", large_countries)
```

11.A.

```
import pandas as pd
import matplotlib.pyplot as plt

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"]
}
df = pd.DataFrame(data)

states_q1 = df[df["Quarter"] == 1]["State"].unique()
print("States in Quarter 1:", states_q1)

employees_q2 = df[df["Quarter"] == 2]["employee"].tolist()
print("Employees in Quarter 2:", employees_q2)
employee_state = df[["employee", "State"]]
print("Employee Names with States:\n", employee_state)

high_sales_employees = df[df["sales"] > 200000][["employee", "sales"]]
print("Employees with Sales above 200000:\n", high_sales_employees)

state_sales = df.groupby("State")["sales"].sum()
```

```
print("State-wise Sales:\n", state_sales)


kerala_avg_sales = df[df["State"] == "Kerala"]["sales"].mean()

higher_than_kerala = df[df["sales"] > kerala_avg_sales][["employee", "sales"]]

print("Employees earning more than Kerala's average sales:\n", higher_than_kerala)

states_with_e = df[df["State"].str.contains("e", case=False, na=False)]["State"].unique()

print("States with 'e' in their name:", states_with_e)

plt.figure(figsize=(8, 5))

plt.bar(df["employee"], df["sales"], color="skyblue")

plt.xlabel("Employee")

plt.ylabel("Sales")

plt.title("Employee vs Sales")

plt.xticks(rotation=45)

plt.show()
```

11.B.

```
class Sample:

    def __init__(self, var):

        self.var = var


    def __sub__(self, other):

        return self.var - other.var

    def __mul__(self, other):

        return self.var * other.var


    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


    def __and__(self, other):
        return self.var & other.var

    def __or__(self, other):

        return self.var | other.var

    # Display method for easy output

    def __str__(self):

        return str(self.var)


a = Sample(10)

b = Sample(5)


print("Subtraction:", a - b)

print("Multiplication:", a * b)

# Logical operations

print("a < b:", a < b)

print("a > b:", a > b)

print("a == b:", a == b)

# Bitwise operations

print("Bitwise AND:", a & b)

print("Bitwise OR:", a | b)
```

12.A.

```
class empbase:

    def get(self):

        self.enum = int(input("Enter your rollno :"))

        self.ename = input("Enter your name :")

        self.basic = int(input("Enter your basic salary :"))


    def put(self):

        print(f"roll No :{self.enum} \nName : {self.ename} \nBasic Salary : {self.basic}\n")


class empchild(empbase):

    def get1(self):
        self.ded = eval(input("Enter the Deduction Amount :"))
        self.allowance = eval(input("Enter the allowance Amount :"))
        self.gross = self.basic + self.allowance
        self.net = self.gross - self.ded

    def put2(self):
        print(f"The value of Deduction is {self.ded}")
        print(f"The value of Allowance is {self.allowance}")
        print(f"The value of gross salary is {self.gross}")
        print(f"The value of net is {self.net}")

    def display(self):
        self.get()
        self.get1()
        print("Details of Employee :")
        print("------------------------")
        self.put()
        self.put2()
        print("------------------------")
ch = empchild()
ch.display()
```

12.B.

```
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["date_of_birth"] = pd.to_datetime(df["date_of_birth"])


largest = df.nlargest(2, "note")
smallest = df.nsmallest(2, "note")
print("Largest Notes:\n", largest)
print("Smallest Notes:\n", smallest)


print("First 2 rows (name & profession):\n", df.loc[:1, ["name", "profession"]])
5
print("Rows with note > 85:\n", df[df["note"] > 85])


engineers = df[df["profession"].str.contains("engineer", case=False)]["name"]
print("Engineers:", engineers.tolist())


j_names = df[df["name"].str.startswith("J")]
print("Persons with names starting with 'J':\n", j_names)


filtered_people = df[(df["profession"] == "Data scientist") | (df["note"] > 90)]
print("Data Scientists or Note > 90:\n", filtered_people)


print("Last 3 rows, 3rd column:\n", df.iloc[-3:, 2])


born_after_2000 = df[df["date_of_birth"].dt.year > 2000]
print("People born after 2000:\n", born_after_2000)
```

13.A.

```
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"]

}

df = pd.DataFrame(data)


states_q1 = df[df["Quarter"] == 1]["State"].unique()

print("States in Quarter 1:", states_q1)


employees_q2 = df[df["Quarter"] == 2]["employee"].tolist()

print("Employees in Quarter 2:", employees_q2)

employee_state = df[["employee", "State"]]

print("Employee Names with States:\n", employee_state)


high_sales_employees = df[df["sales"] > 200000][["employee", "sales"]]

print("Employees with Sales above 200000:\n", high_sales_employees)


state_sales = df.groupby("State")["sales"].sum()

print("State-wise Sales:\n", state_sales)


kerala_avg_sales = df[df["State"] == "Kerala"]["sales"].mean()

higher_than_kerala = df[df["sales"] > kerala_avg_sales][["employee", "sales"]]

print("Employees earning more than Kerala's average sales:\n", higher_than_kerala)

quarter_wise = df.groupby('Quarter')['sales'].agg(['mean','median',max,min,])

print(quarter_wise)

print("the total number of unique values :",df['employee'].unique().size)
```

13.B.

```
class Employee:


    company_name = "Tech Corp"


    def __init__(self, name, salary):

        """Constructor: Initializes object variables"""

        self.name = name
        self.salary = salary
        print(f"Employee {self.name} is created.")


    def display_info(self):
        """Member function: Displays employee details"""
        print(f"Name: {self.name}, Salary: {self.salary}, Company: {Employee.company_name}")
    def __del__(self):
        """Destructor: Called when an object is deleted"""
        print(f"Employee {self.name} is deleted.")
emp1 = Employee("Alice", 50000)
emp2 = Employee("Bob", 60000)


emp1.display_info()
emp2.display_info()


print(f"Company Name: {Employee.company_name}")


del emp1
```

14.A.

```
class base:
    def get(self):
        self.num1 = eval(input("Enter the number 1 value :"))
        self.num2 = eval(input("Enter the number 2 value :"))


    def put(self):
        print(f"The num1 is {self.num1} \nThe num2 is {self.num2}")


class child1(base):
    def arithmetic(self):
        self.add = self.num1 + self.num2
        self.sub = self.num1 - self.num2
        self.mul = self.num1 * self.num2
        self.div = self.num1 / self.num2


    def put1(self):
        self.get()
        self.arithmetic()
        print(f"Addition is {self.add} \nSubtraction is {self.sub} \nMultiplication is {self.mul} \nDivision is {self.div}")


class child2(base):
    def logical(self):
        self.andOper = (self.num1 > self.num2) and (self.num1 < self.num2)
        self.orOper = (self.num1 < self.num2) or (self.num1 > self.num2)
        self.notOper = not(self.num1 > self.num2)


    def put2(self):
        print("Values For Logical Operation")
        self.get()
        self.logical()
        print(f"(self.num1 > self.num2) and (self.num1 < self.num2) is {self.andOper} \n(self.num1 < self.num2) or (self.num1 > self.num2) is {self.orOper} \n not(self.num1 > self.num2) is {self.notOper}")


obj1 = child1()
obj1.put1()
obj2 = child2()
obj2.put2()
```

output:

```
10
5
7
15
```

14.B.

```
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["date_of_birth"] = pd.to_datetime(df["date_of_birth"])

largest_smallest = pd.concat([df.nlargest(2, "note"), df.nsmallest(2, "note")])
print("\nLargest and Smallest based on note:\n", largest_smallest)

print("\nFirst 2 rows (name and note):\n", df.loc[:1, ["name", "note"]])
print("\nRows with note > 90:\n", df[df["note"] > 90])

print("\nNames of engineers:\n", df[df["profession"].str.contains("engineer", case=False)]["name"])
print("\nPersons whose name starts with 'J':\n", df[df["name"].str.startswith("J")])

print("\nData scientists or note > 90:\n", df[(df["profession"] == "Data scientist") | (df["note"] > 90)])

print("\nLast 3 rows - Third column (Profession):\n", df.iloc[-3:, 2])

print("\nNames in group A or C:\n", df[df["group"].isin(["A", "C"])]["name"])

print("\nNames of athletes:\n", df[df["profession"] == "Athlete"]["name"])

print("\nPersons born after 2000:\n", df[df["date_of_birth"].dt.year > 2000])
```

15.A.

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

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

    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

    def __and__(self, other):
        return self.var & other.var
    def __or__(self, other):
        return self.var | other.var
    # Display method for easy output
    def __str__(self):
        return str(self.var)

a = Sample(10)
b = Sample(5)

print("Subtraction:", a - b)
print("Multiplication:", a * b)
# Logical operations
print("a < b:", a < b)
print("a > b:", a > b)
print("a == b:", a == b)
# Bitwise operations
```

```
print("Bitwise AND:", a & b)

print("Bitwise OR:", a | b)
```

15.B.

```
Deepak = {'Python' , 'Java' , 'C','C++'}

uma = {'PHP','SQL','ASP.NET','C'}

print(f'Deepak : {Deepak} \nUma :{uma}')

inte = Deepak.intersection(uma)

print("The commom between Deepak and Uma :",inte)

print("Languages known by both of them :",Deepak.union(uma))

print("Languages known by Deepak not by Uma",Deepak.difference(uma))

print("Languages known by Uma not by Deepak",uma.difference(Deepak))

Deepak.add('Go')

print(f"After update Deepak is {Deepak}")

uma.remove('SQL')

print(f"After removing a language in Uma is {uma}")
```

16.A.

```
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", None, "yellow", None, "black", "green", "red"]

}

df = pd.DataFrame(data)


grouped_colors = df.groupby("city")["favourite_color"].apply(lambda x: list(x.dropna()))

print("\nGrouped colors by city:\n", grouped_colors)


cities_ending_with_i = df[df["city"].str.endswith("i")]["city"].unique()
```

```
print("\nCities ending with 'i':\n", cities_ending_with_i)


null_values = df.isnull().sum()

print("\nNull values per column:\n", null_values)


unique_cities = df["city"].unique()

print("\nUnique city names:\n", unique_cities)


gender_count = df["gender"].value_counts()

print("\nCount of males and females:\n", gender_count)


average_age_by_city = df.groupby("city")["age"].mean()

print("\nAverage age per city:\n", average_age_by_city)


total_age_by_section = df.groupby("section")["age"].sum()

print("\nTotal age per section:\n", total_age_by_section)


count_per_city = df["city"].value_counts()

print("\nNumber of persons in each city:\n", count_per_city)
```

16.B.

```
def divisible_by_3(mylist):
    for i in mylist:
        if i%3 == 0:
            l1.append(i)
        else:
            l2.append(i)
mylist =[11, 2, 31, 23, 12, 8, 9, 13, 5, 37]
for i in range(10):
    mylist.append(int(input(f"Enter {i} number in the list :")))


print("The full List :" , mylist)
mylist.sort()
```

```
print("Ascending order :" , mylist)
mylist.sort(reverse=True)
print("Descending order :" , mylist)
l1 = []
l2 =[]
divisible_by_3(mylist)
print(f"Divisible by 3 :{l1}")
print(f"Not Divisible by 3 :{l2}")
l1.remove(max(l1))
l2.remove(min(l2))
print("After removing biggest number from L1 :",l1)
print("After removing smallest number from L2 :",l2)
```

17.A.

```
import numpy as np
a = np.arange(40,50)
print("The Array:",a,"\nAnd its type :", type(a))
print("Size of array:",a.size)
print("Dimension of array:",a.ndim)
print("Sum of elements:", np.sum(a))
print("Mean of elements:", np.mean(a))
print("Minimum value:", np.min(a))
print("Maximum value:", np.max(a))
```

17.B.

```
import pandas as pd
import matplotlib.pyplot as plt


data = {
    "Name": ["Asha", "Harsh", "Sourav", "Hritik", "Shivansh", "Akash", "Soumya", "Kartik"],
    "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],
  "Exp": [10, 2, 3, 4, 7, 3, 2, 8]
}
df = pd.DataFrame(data)

print("Number of employees in each department:")
print(df["Dept"].value_counts())

print("\nPart-time employees in the Marketing department:")
print(df[(df["Dept"] == "Marketing") & (df["Type"] == "Parttime")])

print("\nAverage and total salary of each department:")
print(df.groupby("Dept")["Salary"].agg(["mean", "sum"]))

print("\nEmployees with experience greater than 2:")
print(df[df["Exp"] > 2])

print("\nSummary info of the dataset:")
print(df.info())

print("\nEmployees grouped by employment type:")
print(df.groupby("Type")["Name"].apply(list))

plt.bar(df["Name"], df["Exp"], color='green')
plt.xlabel("Employee Name")
plt.ylabel("Experience (years)")
plt.title("Employee Experience")
plt.xticks(rotation=45)
plt.show()

least_salaried = df.loc[df["Salary"].idxmin()]
print("\nLeast salaried person:")
print(least_salaried)
```

18.A.

```
file = open('Inventory' , 'w+')

file.write("The products are :\n")

for i in range(5):

    print(f"Enter the product details for {i+1}")

    pnum = input("Enter the product number :")

    pname = input("Enter the product name :")

    unit_price = float(input("Enter Unit Price: "))

    quantity = int(input("Enter Quantity: "))

    amount = unit_price * quantity


    file.write(f"pnum :{pnum}\npname : {pname}\nunit_price :{unit_price}\nquantity : {quantity}\nAmount :{amount}\n")

print("File is written successfully")

file.close()

file = open("Inventory" ,'r+')

contents = file.read()

print('\nFile data are')

print('\n-------------------------------------------------------')

print(contents)

print('\n-------------------------------------------------------')

print('\n the file contents is successfully retrived')

file.close()
```

output:

put 5 items

18.B.

```
Deepak = {'Python' , 'Java' , 'C','C++'}

uma = {'PHP','SQL','ASP.NET','C'}

print(f'Deepak : {Deepak} \nUma :{uma}')

inte = Deepak.intersection(uma)

print("The commom between Deepak and Uma :",inte)

print("Languages known by both of them :",Deepak.union(uma))
print("Languages known by Deepak not by Uma",Deepak.difference(uma))

print("Languages known by Uma not by Deepak",uma.difference(Deepak))

Deepak.add('Go')

print(f"After update Deepak is {Deepak}")

uma.remove('SQL')

print(f"After removing a language in Uma is {uma}")
```

19.A.

```
import pandas as pd

import matplotlib.pyplot as plt

data = {

   "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],  # in million sq. km

   "population": [200.4, 143.5, 1252, 1357, 52.98]  # in million

}

df = pd.DataFrame(data)

df_sorted = df.sort_values(by="population", ascending=False)

print("Countries sorted by population:\n", df_sorted)


plt.figure(figsize=(6,6))

plt.pie(df['area'], labels=df['country'], autopct='%1.1f%%', startangle=140)

plt.title("Area occupied by each country")

plt.show()

plt.figure(figsize=(8,5))

plt.bar(df['country'], df['population'], color='yellow')

plt.xlabel("Country")

plt.ylabel("Population (millions)")

plt.title("Population of Each Country")

plt.show()

print("Country and Population:\n", df[['country', 'population']])

capital_china = df[df["country"] == "China"]["capital"].values[0]

print("Capital of China:", capital_china)
```

```
capitals_ending_a = df[df["capital"].str.endswith("a")]["capital"]
print("Capitals ending with 'a':\n", capitals_ending_a.to_list())
null_values = df.isnull().sum()
print("Number of null values in each column:\n", null_values)
smallest_country = df[df["area"] == df["area"].min()]["country"].values[0]
print("Smallest country by area:", smallest_country)
```

19.B.

```
def check_pal(s):
    a = s[::-1]
    if a == s:
        return True
    else:
        return False
def char_count(txt="hello"):
    lower = upper = space = special = number = 0
    for i in txt:
        if i.isupper():
            upper += 1
        elif(i.islower()):
            lower += 1
        elif(i.isspace()):
            space +=1
        elif(i.isdigit()):
            number +=1
        else:
            special +=1
    print(f"This text contains lowerCase : {lower} \nUpperCase :{upper} \nBlankSpace : {space} \nSpecial Char : {special} \nNumbers : {number}")
def count_the(txt='Hellllo the world'):
    a = txt.split()
    c=a.count('the')
    print(f"\nThis text contains {c} 'the' \n")
s = input("Enter a string :")
```

```
s1 = input("Enter another string :")
print('----------------------------------------------------------------')
print(f'\nThe length of string {s}',len(s))
print('\n checking palindrome :')
if(check_pal(s)):
    print("The given string is palindrome \n")
else:
    print("Not a palindrome \n")
print(f"Checking two string {s} and {s1} is equal or not ", s == s1 ,"\n")
para = ''' 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) '''
char_count(para)
count_the(para)
print(para.swapcase())
```

output:

```
Enter a string :madam
Enter another string :hello
```

20.A.

```
def prime(n):
    for i in range(1,n+1):
        if i < 2:
            print(f"The number {i} is not a prime number")
            continue
        flag = 0
        for j in range(2,i):
            if i % j == 0:
                flag =1
                break
        if(flag == 1):
            print(f"The number {i} is not a prime number")
        else:
            print(f'The number {i} is a prime number')
```

```
try:
    n = int(input("Enter a number :"))
    prime(n)
except ValueError :
    print("The value of n must be integer but provided another datatype")
```

20.B.

```
import pandas as pd
import numpy as np
data = {
    'Courses': ["Spark", "PySpark", "Hadoop", "Python", "Pandas", np.nan, "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]
}
df = pd.DataFrame(data)
print(f"Number of rows and columns: {df.shape}")
print("\nColumn names and data types:")
print(df.dtypes)
print("\nCourses and Duration for 1,3,5 rows:")
print(df.loc[[1, 3, 5], ['Courses', 'Duration']])
print("\nCourses with discount between 1000 and 2000:")
print(df[df['Discount'].between(1000, 2000)][['Courses', 'Discount']])
df["Tutors"] = ['William', 'Henry', 'Michael', 'John', 'Messi', 'Ramana', 'Kumar', 'Vasu']
print("\nDataFrame after adding Tutors column:")
print(df)
print("\nNull values statistics:")
print(df.isnull().sum())
print("\nCourses that start with 'P':")
print(df[df['Courses'].str.startswith("P", na=False)])
print("\nCourses with Duration more than 40 days:")
print(df[df['Duration'] > 40][['Courses', 'Duration']])
```