

Sr. No.	Practical No.	Date	Sign.
1	Practical No. 1	08-02-2025	
2	Practical No. 2	10-02-2025	
3	Practical No. 3	03-03-2025	
4	Practical No. 4	05-03-2025	
5	Practical No. 5	10-03-2025	
6	Practical No. 6	12-03-2025	
7	Practical No. 7	17-03-2025	
8	Practical No. 8	19-03-2025	
9	Practical No. 9	24-03-2025	
10	Practical No. 10	26-03-2025	
11	Practical No. 11	09-04-2025	
12	Practical No.12	16-04-2025	
13	Practical No. 13	30-04-2025	
14			
15			

Practical No 1

1. Write a Python program that calculates the area of a circle based on the radius entered by the user.

```
import math
radius = float(input("Enter the radius of the circle: ")) # Taking input from the user
area = math.pi * (radius ** 2) # Formula for area of circle:  $\pi r^2$ 
print(f"r = {radius} Area = {area}") # Displaying the result
```

OUTPUT

```
Enter the radius of the circle: 5
r = 5.0 Area = 78.53981633974483
```

2. Write a Python program that accepts the user's first and last name and prints them in reverse order with a space between them.

```
first_name = input("Enter your first name: ") # Taking first name as input
last_name = input("Enter your last name: ") # Taking last name as input
print(last_name + " " + first_name) # Printing names in reverse order
```

OUTPUT

```
Enter your first name: Yash
Enter your last name: jawarkar
Yash jawarkar
```

3. Write a Python program that accepts a sequence of comma-separated numbers from the user and generates a list and a tuple of those numbers.

```
numbers = input("Enter comma-separated numbers: ") # Taking input as a string
num_list = numbers.split(",") # Splitting string into list elements
num_tuple = tuple(num_list) # Converting list to tuple
print("List:", num_list) # Printing list
print("Tuple:", num_tuple) # Printing tuple
```

OUTPUT

```
Enter comma-separated numbers: 1,2,3,4
List: ['1', '2', '3', '4']
Tuple: ('1', '2', '3', '4')
```

4. Write a Python program that determines whether a given number (accepted from the user) is even or odd, and prints an appropriate message to the user.

```
num = int(input("Enter a number: ")) # Taking number input
if num % 2 == 0: # Checking divisibility by 2
    print(f"{num} is an Even number")
else:
    print(f"{num} is an Odd number")
```

OUTPUT

```
Enter a number: 7
7 is an Odd number
Enter a number: 8
8 is an Even number
```

5. Write a Python program to concatenate N strings.

```
n = int(input("Enter the number of strings you want to concatenate: ")) # Taking count of strings
concatenated_string = "" # Initializing empty string
for _ in range(n): # Loop runs n times
    user_string = input("Enter a string: ") # Taking input string concatenated_string += user_string #
Concatenating strings
print("Concatenated String:", concatenated_string) # Printing the final concatenated result
```

OUTPUT

Enter the number of strings you want to concatenate: 3 Enter a string: Hello

Enter a string:

Enter a string: World! Concatenated String: Hello World!

Practical No 2

1. Write a Python program to do arithmetical operations addition and division.

```
a = float(input("Enter first number: "))
b = float(input("Enter second number: ")) addition = a + B
division = a / b if b != 0 else "Undefined (division by zero)" print("Addition:", addition)
print("Division:", division)
```

OUTPUT

```
Enter first number: 10 Enter second number:
2 Addition: 12.0
Division: 5.0
```

2. Write a Python program to find the area of a triangle.

```
base = float(input("Enter base of the triangle: ")) height = float(input("Enter height of the triangle:
"))
area = 0.5 * base * height # Formula for area of triangle: (1/2) * base * height print("Area of the
triangle:", area)
```

OUTPUT

```
Enter first variable: Hello Enter second variable:
World
After swapping: x = World y = Hello
```

3. Write a Python program to swap two variables.

```
x = input("Enter first variable: ") y = input("Enter second variable:
") x, y = y, x
print("After swapping: x =", x, "y =", y)
```

OUTPUT

```
Enter base of the triangle:
5 Enter height of the triangle: 4 Area of the triangle: 10.0
```

4. Write a Python program to generate a random number.

```
import random
rand_num = random.randint(1, 100) print("Random number:",
rand_num)
```

OUTPUT

```
Random number: 57
```

5. Write a Python program to convert kilometers to miles.

```
km = float(input("Enter distance in kilometers: ")) miles = km * 0.621371
print(f'{km} kilometers is equal to {miles} miles')
```

OUTPUT

```
Enter distance in kilometers: 10
10.0 kilometers is equal to 6.21371 miles
```

6. Write a Python program to display calendar.

```
import calendar
year = int(input("Enter year: "))
month = int(input("Enter month (1-12): "))
print(calendar.month(year, month))
```

OUTPUT

```
Enter year: 2023
Enter month (1-12): 3
March 2023 Mo Tu We Th Fr Sa
Su 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

7. Write a Python program to swap two variables without temp variable.

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
a = a + b
b = a - b
a = a - b
print("After swapping: a =", a, "b =", b)
```

OUTPUT

```
Enter first number: 15 Enter second number:
25
After swapping: a = 25 b = 15
```

8. Write a Python program to check if a number is positive, negative, or zero.

```
num = float(input("Enter a number: "))
```

```
")) if num > 0: print("The number is Positive") elif num < 0: print("The number is Negative") else:  
print("The number is Zero")
```

OUTPUT

Enter a number: -7

The number is Negative

9. Write a Python program to check if a year is a leap year.

```
year = int(input("Enter a year: "))  
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0): print(year, "is a Leap Year")  
else:  
print(year, "is not a Leap Year")
```

OUTPUT

Enter a year: 2020 2020 is a Leap Year

10. Write a Python program to check if a number is odd or even.

```
num = int(input("Enter a number: ")) if num % 2 == 0:  
    print(f"{num} is Even") else:  
print(f"{num} is Odd")
```

OUTPUT

Enter a number: 4 4 is Even

Practical No 3

1. Create a dictionary of your favourite books and their authors and print it.

```
books = {  
    "1984": "George Orwell",  
    "To Kill a Mockingbird": "Harper Lee", "The Great Gatsby": "F. Scott Fitzgerald", "Pride and Prejudice":  
    "Jane Austen"  
}  
print("Favorite Books and Authors:", books)
```

OUTPUT

Favorite Books and Authors: {'1984': 'George Orwell', 'To Kill a Mockingbird': 'Harper Lee', 'The Great Gatsby': 'F. Scott Fitzgerald', 'Pride and Prejudice': 'Jane Austen'}

2. Add a new book to the dictionary and print the updated dictionary.

```
books["The Catcher in the Rye"] = "J.D. Salinger" # Adding a new book  
print("Updated Books Dictionary:", books)
```

OUTPUT

Updated Books Dictionary: {'1984': 'George Orwell', 'To Kill a Mockingbird': 'Harper Lee', 'The Great Gatsby': 'F. Scott Fitzgerald', 'Pride and Prejudice': 'Jane Austen', 'The Catcher in the Rye': 'J.D. Salinger'}

3. Remove a book from the dictionary and print the updated dictionary.

```
books.pop("1984") # Removing a book by title  
print("Dictionary after removing a book:", books)
```

OUTPUT

Dictionary after removing a book: {'To Kill a Mockingbird': 'Harper Lee', 'The Great Gatsby': 'F. Scott Fitzgerald', 'Pride and Prejudice': 'Jane Austen', 'The Catcher in the Rye': 'J.D. Salinger'}

4. Use the keys() method to print a list of the book titles in the dictionary.

```
print("Book Titles:", list(books.keys())) # Extracting only book titles
```

OUTPUT

Book Titles: ['To Kill a Mockingbird', 'The Great Gatsby', 'Pride and Prejudice', 'The Catcher in the Rye']

5. Use the values() method to print a list of the author names in the dictionary.

```
print("Author Names:", list(books.values())) # Extracting only author names
```

Author Names: ['Harper Lee', 'F. Scott Fitzgerald', 'Jane Austen', 'J.D. Salinger']

6. Create a set of your favourite colours and print it.

```
colors = {"Red", "Blue", "Green", "Black", "White"}  
print("Favorite Colors Set:", colors)
```

OUTPUT

Favorite Colors Set: {'Blue', 'Green', 'White', 'Black', 'Red'}

7.Add a new colour to the set and print the updated set.

```
colors.add("Yellow") # Adding a new color print("Updated Colors Set:", colors) OUTPUT
```

Updated Colors Set: {'Blue', 'Green', 'White', 'Black', 'Red', 'Yellow'}

8.Remove a colour from the set and print the updated set.

```
colors.discard("White") # Removing a color print("Set after removing a color:", colors) OUTPUT
```

Set after removing a color:

{'Blue', 'Green', 'Black', 'Red', 'Yellow'}

9.Create a new set that contains only the colours that start with the letter "B" and print it.

```
colors_starting_with_B = {color for color in colors if color.startswith("B")} print("Colors starting with 'B':", colors_starting_with_B)
```

OUTPUT

Colors starting with 'B': {'Blue', 'Black'}

10.Use the len() function to find the number of colours in the set and print it.

```
print("Number of colors in the set:", len(colors))
```

OUTPUT

Number of colors in the set: 5

Practical No 4

1. Program to Find the GCD of Two Positive Numbers.

```
num1 = int(input("Enter first positive number: ")) num2 = int(input("Enter second positive number: ")) while num2: num1, num2 = num2, num1 % num2 print(f"GCD of the numbers is {num1}")
```

OUTPUT

Enter first positive number: 48 Enter second positive number: 18 GCD of the numbers is 6

2. Write Python Program to Find the Sum of Digits in a Number.

```
num = int(input("Enter a number: ")) sum_of_digits = 0 while num: sum_of_digits += num % 10 num //= 10 print(f"Sum of digits is {sum_of_digits}")
```

OUTPUT

Enter a number: 1234 Sum of digits is 10

3. Write a program that prints the first 10 multiples of 3.

```
multiples_of_3 = [3 * i for i in range(1, 11)] print("First 10 multiples of 3:", multiples_of_3) OUTPUT First 10 multiples of 3: [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

LIST

1. Create a list of your favourite Hindi comedy movies and print the third movie in the list.

```
movies = ["Hera Pheri", "Andaz Apna Apna", "Dhamaal", "Chup Chup Ke", "Golmaal: Fun Unlimited"] print("Third movie in the list:", movies[2])
```

OUTPUT

Third movie in the list: Dhamaal

2. Add a new movie to the list and print the updated list.

```
movies.append("Bhool Bhulaiyaa") print("Updated Movie List:", movies)
```

OUTPUT

Updated Movie List: ['Hera Pheri', 'Andaz Apna Apna', 'Dhamaal', 'Chup Chup Ke', 'Golmaal: Fun Unlimited', 'Bhool Bhulaiyaa']

3. Remove the second movie from the list and print the updated list.

```
movies.pop(1)
print("List after removing the second movie:", movies)
```

OUTPUT

List after removing the second movie: ['Hera Pheri', 'Dhamaal', 'Chup Chup Ke', 'Golmaal: Fun Unlimited', 'Bhool Bhulaiyaa']

4. Sort the list in alphabetical order and print the sorted list.

```
movies.sort()
print("Sorted Movie List:", movies)
```

OUTPUT

Sorted Movie List: ['Bhool Bhulaiyaa', 'Chup Chup Ke', 'Dhamaal', 'Golmaal: Fun Unlimited', 'Hera Pheri']

5. Create a new list that contains only the first and last movie in the original list and print it.

```
first_last_movies = [movies[0], movies[-1]] print("First and Last Movie:", first_last_movies) OUTPUT
First and Last Movie: ['Hera Pheri', 'Bhool Bhulaiyaa']
```

TUPLE

1. Create a tuple of your favourite foods and print the second food in the tuple.

```
foods = ("Pizza", "Biryani", "Pani Puri", "Chole Bhature", "Dosa") print("Second food in the tuple:", foods[1])
```

OUTPUT

Second food in the tuple: Biryani

2. Try to change the second food in the tuple and see what happens.

```
try:
    foods[1] = "Pasta" # Tuples are immutable, so this will raise an error except TypeError as e:
    print("Error:", e)
```

OUTPUT

Error: 'tuple' object does not support item assignment

3. Create a new tuple that contains only the first and last foods in the original tuple and print it.

```
first_last_foods = (foods[0], foods[-1]) print("First and Last Food:", first_last_foods) OUTPUT
First and Last Food: ('Pizza', 'Dosa')
```

4. Use the len() function to find the number of foods in the tuple and print it.

```
print("Number of foods in the tuple:", len(foods))
```

OUTPUT

Number of foods in the tuple: 5

5. Convert the tuple to a list and print the list.

```
foods_list = list(foods)  
print("Tuple converted to list:", foods_list)
```

OUTPUT

Tuple converted to list: ['Pizza', 'Biryani', 'Pani Puri', 'Chole Bhature', 'Dosa']

Practical No 5

1. Print all even numbers from 0 to the given number

```
num = int(input("Enter a number: ")) i = 0
while i <= num:
    if i % 2 == 0: print(i)
    i += 1
```

OUTPUT

Enter a number: 10

*0
2
4
6
8
10*

2. Print each character of a string on a new line

```
text = input("Enter a string: ") for char in text:
    print(char)
```

OUTPUT

Enter a string: Hello
*He
l
l
o*

3. Print pattern

```
a) for i in range(1, 6): for j in range(1, i + 1):
    print(j, end=" ") print()
```

```
b) ch = 65
for i in range(1, 5): for j in range(i):
    print(chr(ch), end=" ") ch += 1
print()
```

c) for i in range(4, 0, -1): for j in range(i):
 print(i, end="") print()

d) for i in range(1, 6, 2): for j in range(5 - i, 0, -1):
 print(" ", end="") for j in range(i):
 print("*", end="")
 print()

e) for i in range(1, 6, 2): for j in range(5 - i, 0, -1):
 print(" ", end="") for j in range(i):
 print("*", end="") print()

f) for i in range(1, 5): for j in range(i):
 print((j + i) % 2, end="") print()

Output

1
 12
 123
 1234
 12345

A
 B C
 C D E
 D E F G

4444
 333
 22
 1

```
*
* * *
* * * * *
```

```
*
***
*****
```

```
1
01
101
0101
```

6. Find ASCII value of a character

```
char = input("Enter a character: ") print("ASCII value of", char, "is", ord(char))
```

OUTPUT

Enter a character: A ASCII value of A is 65

7. Simple calculator

```
a = float(input("Enter first number: "))
b = float(input("Enter second number: ")) op = input("Enter operation (+, -, *, /): ") if op == "+":
    print("Result:", a + b) elif op == "-":
    print("Result:", a - b) elif op == "*":
    print("Result:", a * b) elif op == "/":
if b != 0: print("Result:", a /
b) else:
    print("Division by zero error") else:
print("Invalid operation")
```

OUTPUT

*Enter first number: 10 Enter second number: 5 Enter operation (+, -, *, /): * Result: 50.0*

8. Find the largest element in an array

```
n = int(input("Enter number of elements: ")) arr = []

for i in range(n): arr.append(int(input()))

largest=arr[0]
for num in arr:
    if num > largest: largest = num
print("Largest element:", largest)
```

OUTPUT

Enter number of elements:

5 1

3

7

2

5

Largest element: 7

9. Add two matrices

```
r = int(input("Enter number of rows: "))
c = int(input("Enter number of columns: ")) mat1 = []
mat2 = [] result = []
for i in range(r):
    row = []
    for j in range(c): row.append(int(input()))
    mat1.append(row)
    row = []
    for j in range(c): row.append(int(input()))
    mat2.append(row)
    row = []
    for j in range(c): row.append(mat1[i][j] + mat2[i][j])
    result.append(row)
```

```
) for row in result:  
for num in row: print(num, end=" ")  
print()
```

OUTPUT

```
Enter number of rows: 2 Enter number of columns: 2 Enter elements of first matrix: 1 2  
3  
4  
Enter elements of second matrix: 5  
6  
7  
8  
Resultant Matrix:  
6 8  
10 12
```


Practical No 6

1. Program to find the sum of all odd and even numbers up to a number specified by the user.

```
def sum_odd_even(n):  
    even_sum = sum(i for i in range(0, n+1, 2)) # Sum of even numbers  
    odd_sum = sum(i for i in range(1, n+1, 2)) # Sum of odd numbers  
    return even_sum, odd_sum
```

```
num = int(input("Enter a number: ")) # Taking input from the user  
even_sum, odd_sum = sum_odd_even(num) # Calling the function
```

```
print(f"Sum of even numbers up to {num} is: {even_sum}")  
print(f"Sum of odd numbers up to {num} is: {odd_sum}")
```

OUTPUT

Enter a number: 10

Sum of even numbers up to 10 is: 30 Sum of odd numbers up to 10 is: 25

2. Program to check if a given string is a palindrome using slicing.

```
def is_palindrome(s):  
    s = s.lower().replace(" ", "") # Converting to lowercase and removing spaces  
    return s == s[::-1] # Checking if the string is equal to its reverse
```

```
user_string = input("Enter a string: ") # Taking input from the user  
if is_palindrome(user_string): #
```

Checking if palindrome

```
    print(f"'{user_string}' is a palindrome.")  
else:  
    print(f"'{user_string}' is not a palindrome.")
```

OUTPUT

Enter a string: Race car 'Race car' is a palindrome.

3. Count Vowels, Consonants, and Blanks in a String

```
def count_chars(s):  
    vowels = "aeiouAEIOU"  
    consonants = sum(1 for c in s if c.isalpha() and c not in vowels)  
    vowel_count = sum(1 for c in s if c in vowels)  
    blanks = s.count(' ')  
    return vowel_count, consonants, blanks
```

OUTPUT

Enter a sentence: Hello World Vowels, consonants, blanks: (3, 7, 1)

4. Print Characters Common in Two Strings

```
def common_chars(s1, s2):  
    return set(s1) & set(s2)
```

OUTPUT

Enter first string: hello

Enter second string: world Common characters: {'l', 'o'}

5. Calculate Percentage of Marks

```
def calculate_percentage(marks): total_marks = sum(marks)
    percentage = (total_marks / (len(marks) * 100)) * 100 return percentage
```

OUTPUT

Enter marks separated by space: 80 90 100

Percentage: 90.0

6. Display Fibonacci Sequence up to n Terms

```
def fibonacci(n): a, b = 0, 1
    fib_seq = []
    for _ in range(n):
        fib_seq.append(a) a, b = b, a + b
    return fib_seq
```

OUTPUT

Enter the number of Fibonacci terms: 7 Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8]

7. Remove Duplicate Words from a Sentence and Sort Them

```
def remove_duplicates_sort(sentence):
    words = list(set(sentence.split())) return " ".join(sorted(words))
```

OUTPUT

Enter a sentence: the quick brown fox jumps over the lazy dog Sorted unique words: brown dog fox jumps lazy over quick the

8. Implement Stack Operations using *args

```
def stack_operations(*args): stack = []
    for op in args:
        if op.startswith("push"):
            val = op.split() stack.append(val)
    elif op == "pop": if stack:
        stack.pop() elif op == "peek":
        if stack:
            return stack[-1] return stack
```

OUTPUT

operations = ["push 10", "push 20", "pop", "peek"] Stack after operations: 10

Practical No 7

1. OOP Concepts in Pharmaceuticals. Using Inheritance, Encapsulation, Abstraction, and Polymorphism

```
from abc import ABC, abstractmethod
class DrugFormulation(ABC):
    def __init__(self, name, dosage, manufacturer):
        self._name = name
        self._dosage = dosage
        self._manufacturer = manufacturer

    @abstractmethod
    def administer(self):
        pass

    def get_info(self):
        return f"{self._name} ({self._dosage}) by {self._manufacturer}"

# Tablet class inherits from DrugFormulation class
class Tablet(DrugFormulation):
    def administer(self):
        return f"Administer {self._name} tablet orally with water."

# Capsule class inherits from DrugFormulation class
class Capsule(DrugFormulation):
    def administer(self):
        return f"Administer {self._name} capsule with warm water."

# Injection class inherits from DrugFormulation class
class Injection(DrugFormulation):
    def administer(self):
        return f"Administer {self._name} injection intravenously."

def prescribe_drug(drug: DrugFormulation):
    return drug.administer()

tablet = Tablet("Paracetamol", "500mg", "PharmaCorp")
capsule = Capsule("Amoxicillin", "250mg", "MediHealth")
injection = Injection("Insulin", "10ml", "BioCare")

print(prescribe_drug(tablet))
print(prescribe_drug(capsule))
print(prescribe_drug(injection))
```

OUTPUT
Administer Paracetamol tablet orally with water.
Administer Amoxicillin capsule with warm water.
Administer Insulin injection intravenously.

2. Program to find the sum of an array

```
def sum_of_array(arr):
    return sum(arr)
```

OUTPUT
arr = [1, 2, 3, 4, 5]
Sum of array: 15

3. Program to find the largest element in an array

```
def largest_element(arr):  
    return max(arr)
```

OUTPUT

```
arr = [1, 2, 3, 4, 5]
```

Largest element in array: 5

4. Program to split the array and add the first part to the end

```
def split_and_add(arr, split_index):  
    return arr[split_index:] + arr[:split_index]
```

OUTPUT

```
arr = [1, 2, 3, 4, 5]
```

```
split_index = 2
```

Array after splitting and adding: [3, 4, 5, 1, 2]

5. Program to add two matrices

```
def add_matrices(matrix1, matrix2):  
    return [[matrix1[i][j] + matrix2[i][j] for j in range(len(matrix1[0]))] for i in range(len(matrix1))]
```

OUTPUT

```
matrix1 = [ [1, 2, 3],  
            [4, 5, 6],  
            [7, 8, 9]  
            ]
```

```
matrix2 = [ [9, 8, 7],  
            [6, 5, 4],  
            [3, 2, 1]  
            ]
```

Sum of matrices: [[10, 10, 10], [10, 10, 10], [10, 10, 10]]

Practical No 8

1. Write a program to create point class with x,y,z coordinate and methods increment point, decrement point, add points, less than, greater than, equal to, check in which quadrant it lies, check whether the point is collinear and print point.

```
class Point:
```

```
    def __init__(self, x, y, z):
```

```
        self.x = x self.y = y self.z = z
```

```
    # Increment each coordinate by 1 def increment(self):
```

```
        self.x += 1
```

```
        self.y += 1
```

```
        self.z += 1
```

```
    def decrement(self):
```

```
        self.x -= 1
```

```
        self.y -= 1
```

```
        self.z -= 1
```

```
    def add(self, other):
```

```
        return Point(self.x + other.x, self.y + other.y, self.z + other.z)
```

```
    def __lt__(self, other):
```

```
        return (self.x, self.y, self.z) < (other.x, other.y, other.z)
```

```
    def __gt__(self, other):
```

```
        return (self.x, self.y, self.z) > (other.x, other.y, other.z)
```

```
    def __eq__(self, other):
```

```
        return (self.x, self.y, self.z) == (other.x, other.y, other.z)
```

```
    def quadrant(self):
```

```
        if self.x > 0 and self.y > 0: return "First Quadrant"
```

```
        elif self.x < 0 and self.y > 0: return "Second Quadrant"
```

```
            elif self.x < 0 and self.y < 0: return "Third Quadrant" elif self.x > 0 and self.y < 0: return "Fourth  
Quadrant"
```

```
        else:
```

```
            return "On Axis"
```

```
def __str__(self):
    return f"Point({self.x}, {self.y}, {self.z})"
```

```
p1 = Point(1, 2, 3)
p2 = Point(-1, -2, 0)
```

```
p1.increment() p2.decrement()
```

```
p3 = p1.add(p2)
```

```
print(str(p1)) print(str(p2)) print(str(p3)) print(p1 < p2) print(p1.quadrant()) print(p2.quadrant())
```

OUTPUT

```
Point(2, 3, 4)
```

```
Point(-2, -3, -1)
```

```
Point(0, 0, 3) False
```

```
First Quadrant Third Quadrant
```

2.Create class watch with hr,min,sec,alarm,type and methods setalarm, stopalarm,showtime.

```
class Watch:
```

```
def __init__(self, hr, min, sec, alarm=None, type="Digital"): self.hr = hr
    self.min = min self.sec = sec self.alarm = alarm self.type = type
```

```
def set_alarm(self, alarm_time): self.alarm = alarm_time
```

```
def stop_alarm(self):
    self.alarm = None
```

```
def show_time(self):
    return f"{self.hr:02}:{self.min:02}:{self.sec:02}"
```

```
watch = Watch(9, 30, 45)
print(watch.show_time()) # Displays time in HH:MM:SS format
```

```
watch.set_alarm("10:00:00") print("Alarm set for:", watch.alarm) watch.stop_alarm()
print("Alarm:", watch.alarm)
```

OUTPUT

09:30:45

Alarm set for: 10:00:00 Alarm: None

3. Write Python Program to Simulate a Bank Account with Support for depositMoney, withdrawMoney and showBalance Operations.

```
class BankAccount:
```

```
def __init__(self, balance=0):
    self.balance = balance
```

```
def deposit_money(self, amount):
    self.balance += amount
```

```
def withdraw_money(self, amount):
    if amount <= self.balance:
        self.balance -= amount
    else:
        print("Insufficient funds!")
```

```
def show_balance(self):
    return f"Balance: {self.balance}"
```

```
account = BankAccount(100) account.deposit_money(50) print(account.show_balance())
account.withdraw_money(75) print(account.show_balance()) account.withdraw_money(100)
print(account.show_balance()) OUTPUT
```

Balance: 150

Balance: 75 Insufficient funds! Balance: 75

4. Create class vehicle with attributes(color,capacity,enginpower,tyre) and behaviour (start,stop) Create class car which inherit vehicle class with

attributes(airbags,gear,speed,fuel,) and methods(accelerate ,fillfuel,playmusic(),onAC()) Create class electric car with attribute(battery) and behaviour(charging(),battery level()).

```
class Vehicle:
    def __init__(self, color, capacity, engine_power, tyres):
        self.color = color
        self.capacity = capacity
        self.engine_power = engine_power
        self.tyres = tyres
```

```
    def start(self):
        print("Vehicle started.")
```

```
    def stop(self):
        print("Vehicle stopped.")
```

```
class Car(Vehicle):
```

```
    def __init__(self, color, capacity, engine_power, tyres, airbags, gear, speed, fuel):
        super().__init__(color, capacity, engine_power, tyres)
        self.airbags = airbags
        self.gear = gear
        self.speed = speed
        self.fuel = fuel
```

```
    def accelerate(self):
        self.speed += 10
```

```
    def fill_fuel(self, amount):
        self.fuel += amount
```

```
    def play_music(self):
        print("Playing music.")
```

```
    def on_ac(self):
        print("AC turned on.")
```

```
class ElectricCar(Car):
```

```
    def __init__(self, color, capacity, engine_power, tyres, airbags, gear, speed, battery):
        super().__init__(color, capacity, engine_power, tyres, airbags, gear, speed, fuel=0)
        self.battery = battery
```

```
    def charging(self):
        print("Car is charging.")
```

```
    def battery_level(self):
```



```
return f"Battery level: {self.battery}%"
```

```
vehicle = Vehicle("Red", 5, "150 HP", 4) vehicle.start()  
vehicle.stop()
```

```
car = Car("Blue", 5, "200 HP", 4, airbags=6, gear="Automatic", speed=50, fuel=10) car.start()  
car.accelerate() car.fill_fuel(20) car.play_music() car.on_ac()  
print(f"Car color: {car.color}, Speed: {car.speed}, Fuel: {car.fuel}")
```

```
e_car = ElectricCar("Green", 5, "180 HP", 4, airbags=4, gear="Manual", speed=40, battery=80)  
e_car.start() e_car.charging() print(e_car.battery_level()) OUTPUT  
Vehicle started. Vehicle stopped. Vehicle started. Playing music. AC turned on.  
Car color: Blue, Speed: 60, Fuel: 30 Vehicle started.  
Car is charging. Battery level: 80%
```

5. Define a class Person and its two child classes: Male and Female. All classes have a method "getGender" which can print "Male" for Male class and "Female" for Female class

```
class Person:  
    def get_gender(self): pass  
  
class Male(Person):  
    def get_gender(self):  
        return "Male"  
  
class Female(Person):  
    def get_gender(self):  
        return "Female"  
  
man = Male() woman = Female()  
print(man.get_gender()) print(woman.get_gender())
```

OUTPUT

Male Female

6.Implement Object-Oriented Programming (OOP) concepts in Python, including Polymorphism, Encapsulation, Inheritance, and Abstraction, using a pharmaceutical- related example

```
from abc import ABC, abstractmethod
class Medicine(ABC):
    @abstractmethod
    def use(self):
        pass
```

```
class Tablet(Medicine):
    def use(self):
        return "Swallow with water."
```

```
class Syrup(Medicine):
    def use(self):
        return "Take with a spoon."
```

```
class Capsule(Medicine):
    def use(self):
        return "Take with warm water."
```

```
class Doctor:
    def prescribe(self, medicine: Medicine):
        return medicine.use()
```

```
doc = Doctor()
tab = Tablet()
print(doc.prescribe(tab))
```

```
doc = Doctor()
tablet_medicine = Tablet()
syrup_medicine = Syrup()
capsule_medicine = Capsule()
```

```
print(doc.prescribe(tablet_medicine))
```

```
print(doc.prescribe(syrup_medicine))
print(doc.prescribe(capsule_medicine))
```

OUTPUT

Swallow with water. Take with a spoon. Take with warm water.

7.Design a Python program to simulate different types of Drug Formulations using OOP principles.

The program should:

- 1. Use Inheritance to create different drug formulations (e.g., Tablet, Capsule, Injection).**
- 2. Implement Encapsulation to protect sensitive drug data.**
- 3. Apply Abstraction to define a blueprint for drug formulations.**
- 4. Demonstrate Polymorphism by overriding methods in different drug types**

```
from abc import ABC, abstractmethod
class DrugFormulation(ABC):
    def __init__(self, name, dosage, manufacturer):
        self._name = name # Encapsulation: Protecting sensitive drug data
        self._dosage = dosage
        self._manufacturer = manufacturer
```

```
@abstractmethod
def administer(self):
    pass
```

```
def get_info(self):
    return f'{self._name} ({self._dosage}) by {self._manufacturer}'
```

```
class Tablet(DrugFormulation):
    def administer(self):
        return f'Administer {self._name} tablet orally with water.'
```

```
class Capsule(DrugFormulation):
    def administer(self):
        return f'Administer {self._name} capsule with warm water.'
```

```
class Injection(DrugFormulation):
    def administer(self):
        return f'Administer {self._name} injection intravenously.'
```

```
def prescribe_drug(drug: DrugFormulation):
    return drug.administer()
```

```
tablet = Tablet("Paracetamol", "500mg", "PharmaCorp")
```

```
capsule = Capsule("Amoxicillin", "250mg", "MediHealth") injection = Injection("Insulin", "10ml",  
"BioCare")
```

```
print(prescribe_drug(tablet)) print(prescribe_drug(capsule)) print(prescribe_drug(injection))
```

OUTPUT

Administer Paracetamol tablet orally with water.

Administer Amoxicillin capsule with warm water.

Administer Insulin injection intravenously

Practical No 9

1. Write a regular expression to extract year, month and date from a string. Extract only 3 digit numbers from string. Extract all of the words and numbers from string. Find out all of the words which start with a vowel. Find out all of the words, which start with a consonant. Count total numbers of a, an and the.

```
import re
```

```
text = "Today's date is 2024-03-24. The event was held on 1999/12/31. Some numbers: 456, 89, Words: apple"
```

```
# 1. Extract Year, Month, and Date from a string
```

```
date_pattern = r"(\d{4})[-/](\d{2})[-/](\d{2})"
```

```
dates = re.findall(date_pattern, text)
```

```
print("Extracted Dates (Year, Month, Day):", dates)
```

```
# 2. Extract Only 3-Digit Numbers
```

```
three_digit_pattern = r"\b\d{3}\b"
```

```
three_digit_numbers = re.findall(three_digit_pattern, text)
```

```
print("Three-digit numbers:", three_digit_numbers)
```

```
# 3. Extract All Words and Numbers
```

```
words_numbers_pattern = r"\w+"
```

```
words_numbers = re.findall(words_numbers_pattern, text)
```

```
print("All words and numbers:", words_numbers)
```

```
# 4. Find All Words That Start with a Vowel
```

```
vowel_pattern = r"\b[aeiouAEIOU]\w*"
```

```
vowel_words = re.findall(vowel_pattern, text)
```

```
print("Words starting with a vowel:", vowel_words)
```

```
# 5. Find All Words That Start with a Consonant
```

```
consonant_pattern = r"\b[^aeiouAEIOU\W]\w*"
```

```
consonant_words = re.findall(consonant_pattern, text)
```

```
print("Words starting with a consonant:", consonant_words)
```

output

```
Extracted Dates (Year, Month, Day): [('2024', '03', '24'), ('1999', '12', '31')] Three-digit numbers: ['456']
All words and numbers: ['Today', 's', 'date', 'is', '2024', '03', '24', 'The', 'event', 'was', 'held', 'on', '1999',
'12', '31', 'Some', 'numbers', '456', '89', 'Words', 'apple']
Words starting with a vowel: ['is', 'event', 'on', 'apple']
Words starting with a consonant: ['Today', 's', 'date', '2024', '03', '24', 'The', 'was', 'held', '1999', '12',
'31', 'Some', 'numbers', '456', '89', 'Words']
```

2. Write a Python program to find all words which are at least 4 characters long in a string.

```
import re
text = "python program to find all words which are at least 4"

# Regular expression to match words with at least 4 characters
pattern = r"\b\w{4,}\b"

# Find all matching words
long_words = re.findall(pattern, text)
print("Words with at least 4 characters:", long_words)
```

output

```
Words with at least 4 characters: ['python', 'program', 'find', 'words', 'which', 'least']
```

3. Write a Python program to check for a number at the end of a string.

```
import re

def ends_with_number(s):
    pattern = r"\d+$"
    return bool(re.search(pattern, s))

test_strings = [
    "Hello123",
    "Python 2024",
]

for text in test_strings:
    print(f'{text} ends with a number? {ends_with_number(text)}')
```

output

'Hello123' ends with a number? True
'Python 2024' ends with a number? True

4. Write a Python program to check for a number starting with 2 or 1 and having 4 digits

```
def check_number(number):
    num_str = str(number)
    if len(num_str) == 4 and (num_str[0] == '1' or num_str[0] == '2'):
        return "Valid: The number starts with 1 or 2 and has 4 digits."
    else:
        return "Invalid: The number doesn't meet the criteria."
number = int(input("Enter a 4-digit number: "))
print(check_number(number))
```

output

Enter a 4-digit number: 1234
Valid: The number starts with 1 or 2 and has 4 digits.

5. Write a Python program that matches a string that has an 'a' followed by anything, ending in 'b'

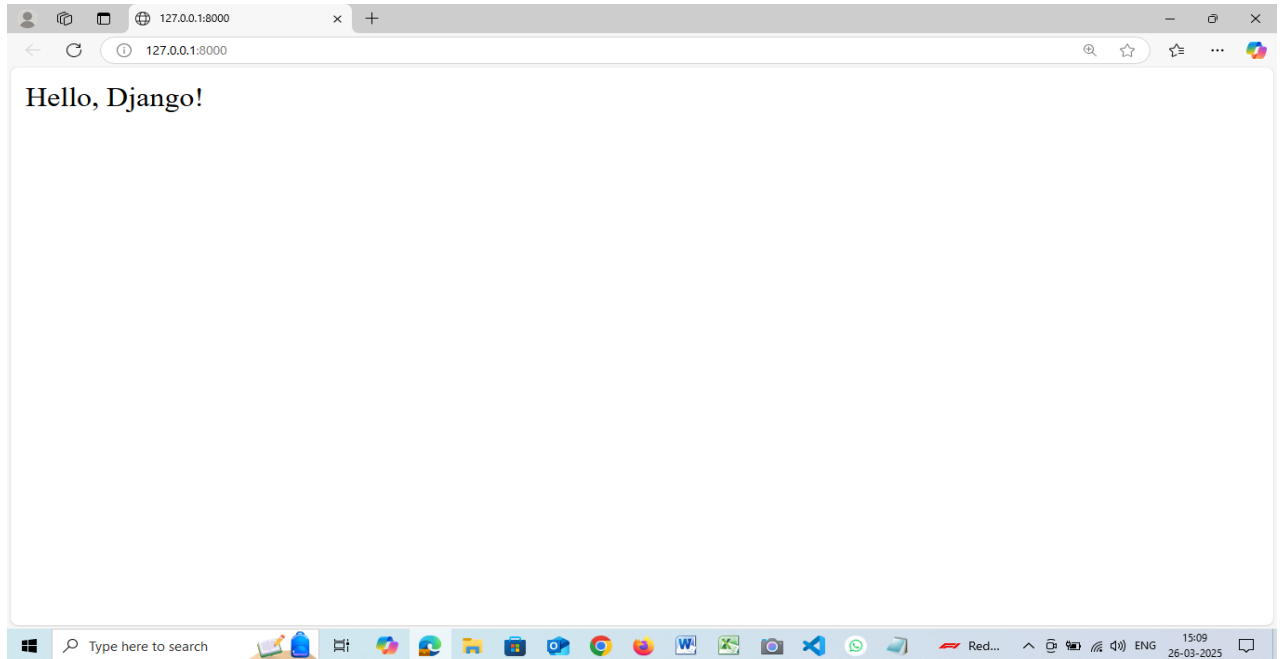
```
import re
def match_string(s):
    pattern = r'^a.*b$' # Regex pattern: 'a' at start (^) followed by anything (.* ) and ending in 'b' ($)
    if re.match(pattern, s):
        return "Valid: The string starts with 'a' and ends with 'b'."
    else:
        return "Invalid: The string does not meet the criteria."
test_str = input("Enter a string: ")
print(match_string(test_str))
```

output

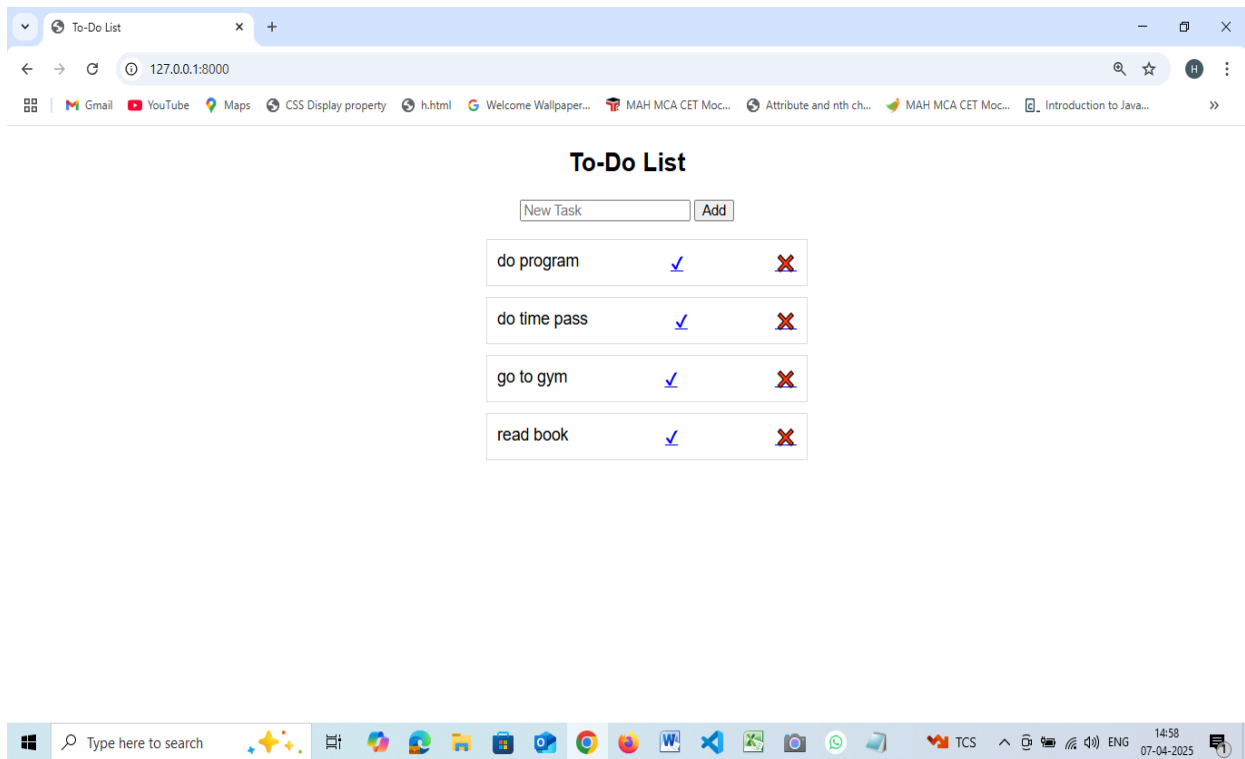
Enter a string: appleb
Valid: The string starts with 'a' and ends with 'b'.

Practical No 10 (Django)

1. Create a simple app that displays "Hello, Django!" in the browser.

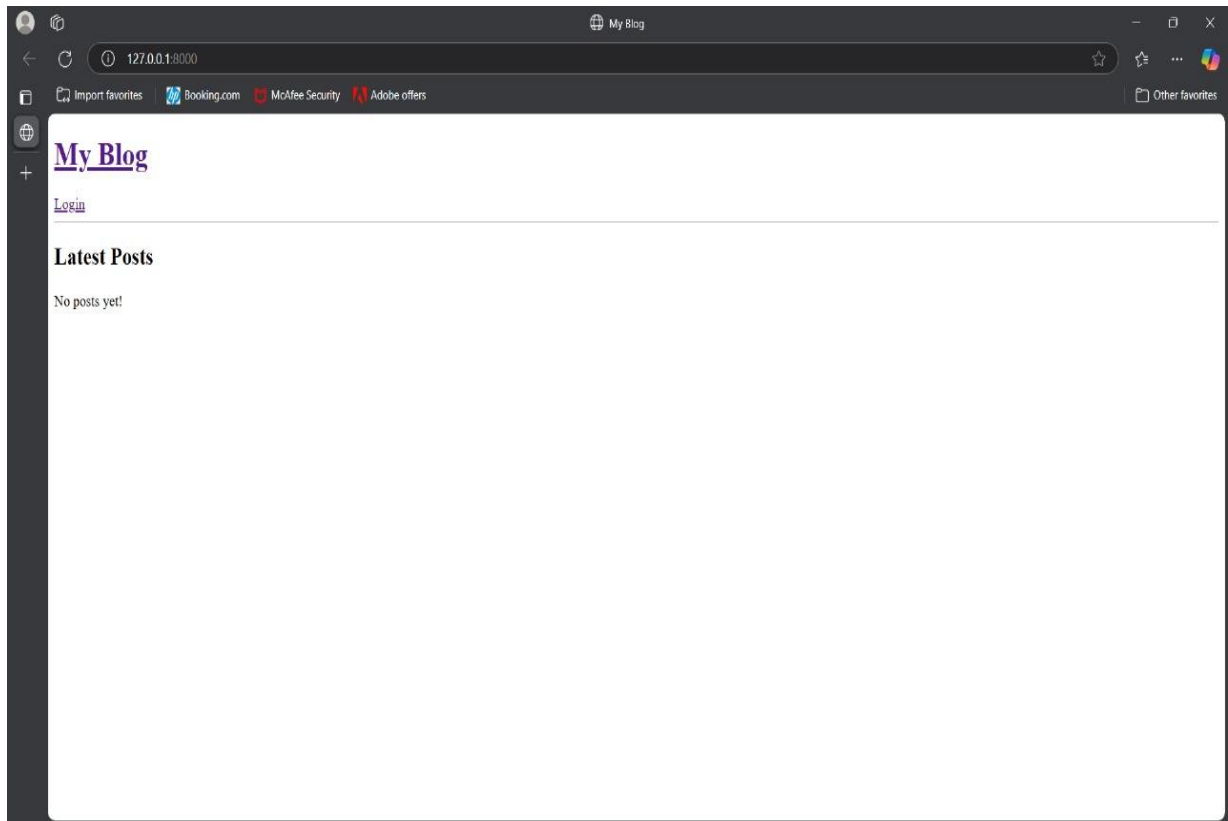


2.Create a basic to-do list app with Django.



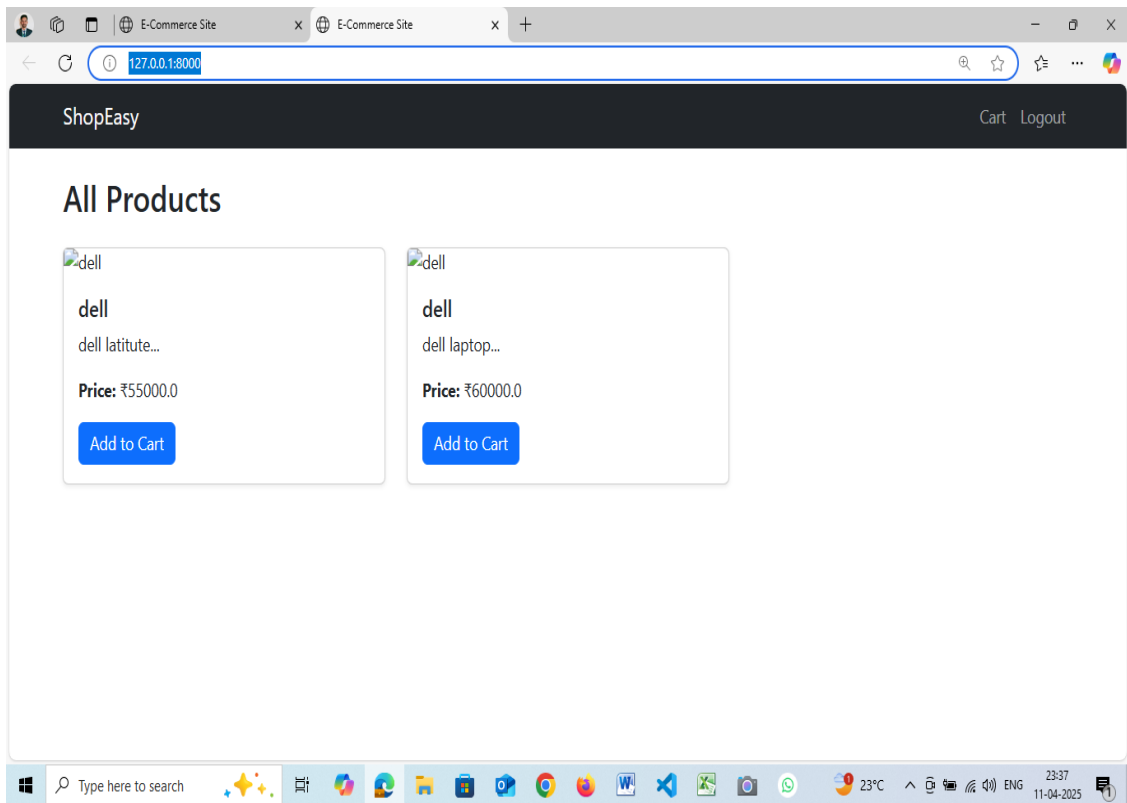
3.Create a Blog Application using Django.

- Users can create, read, update, and delete blog posts.
- Features like categories, comments, and user authentication.
- Example: A personal blogging platform like WordPress.



4.E-Commerce Website using Django.

- Product catalog, shopping cart, and payment integration.
- Order management, user profiles, and review systems.
- Example: Online stores like Amazon or Shopify.



Practical No 11

1.Insert Single Document – A program to insert one document into a MongoDB collection. from pymongo import MongoClient

```
# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Create/Get a database
db = client["mydatabase"]

# Step 3: Create/Get a collection
collection = db["students"]

# Step 4: Create a document (Python dictionary)
student_data = {
    "name": "Yash Jawarkar",
    "age": 23,
    "course": "MCA",
    "city": "Pune"
}

# Step 5: Insert the document
result = collection.insert_one(student_data)

# Step 6: Print inserted ID and success message
print("Document inserted with ID:", result.inserted_id)
```

output

Document inserted with ID: 66191b8b1b3c23b157f5db0e

2.Insert Multiple Documents – Inserts a list of documents into a collection at once.

```
from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Create/Get a database
db = client["mydatabase"]

# Step 3: Create/Get a collection
collection = db["students"]

# Step 4: Create a list of documents (list of dictionaries)
students_data = [
    {"name": "Rahul", "age": 22, "course": "BCA", "city": "Mumbai"},
```

```

{"name": "Sneha", "age": 24, "course": "MCA", "city": "Pune"},
{"name": "Amit", "age": 23, "course": "MBA", "city": "Delhi"},
{"name": "Priya", "age": 21, "course": "BBA", "city": "Chennai"}
]

```

```

# Step 5: Insert the documents
result = collection.insert_many(students_data)

```

```

# Step 6: Print inserted IDs
print("Documents inserted with IDs:")
for doc_id in result.inserted_ids:
    print(doc_id)

```

output

```

Documents inserted with IDs:
66191c5a1b3c23b157f5db0f
66191c5a1b3c23b157f5db10
66191c5a1b3c23b157f5db11
66191c5a1b3c23b157f5db12

```

3.Find All Documents – Retrieves and prints all documents from a specified collection.

```

from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Find all documents in the collection
all_documents = collection.find()

# Step 5: Print all documents
print("All Documents in 'students' collection:")
for doc in all_documents:
    print(doc)

```

output

```

All Documents in 'students' collection:
{'_id': ObjectId('66191b8b1b3c23b157f5db0e'), 'name': 'Yash Jawarkar', 'age': 23, 'course': 'MCA',
'city': 'Pune'}

```

```
{'_id': ObjectId('66191c5a1b3c23b157f5db0f'), 'name': 'Rahul', 'age': 22, 'course': 'BCA', 'city': 'Mumbai'}
{'_id': ObjectId('66191c5a1b3c23b157f5db10'), 'name': 'Sneha', 'age': 24, 'course': 'MCA', 'city': 'Pune'}
{'_id': ObjectId('66191c5a1b3c23b157f5db11'), 'name': 'Amit', 'age': 23, 'course': 'MBA', 'city': 'Delhi'}
{'_id': ObjectId('66191c5a1b3c23b157f5db12'), 'name': 'Priya', 'age': 21, 'course': 'BBA', 'city': 'Chennai'}
```

4.Find Document by Field – Searches for documents that match a specific field value (e.g., name = "Alice").

```
from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Define the query
query = {"name": "Sneha"}

# Step 5: Search and print matching documents
results = collection.find(query)

print("Documents where name = 'Sneha':")
for doc in results:
    print(doc)
```

output

```
Documents where name = 'Sneha':
{'_id': ObjectId('66191c5a1b3c23b157f5db10'), 'name': 'Sneha', 'age': 22, 'course': 'MCA', 'city': 'Pune'}
```

5.Update a Single Document – Updates the first document that matches a given condition.

```
from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]
```

```

# Step 3: Select the collection
collection = db["students"]

# Step 4: Define the query and the new values
query = {"name": "Sneha"} # Condition
new_values = {"$set": {"city": "Bangalore"}} # Update operation

# Step 5: Perform the update
result = collection.update_one(query, new_values)

# Step 6: Output result
if result.modified_count > 0:
    print("Document updated successfully.")
else:
    print("No document matched or update was not necessary.")

```

output

Document updated successfully.

6.Update Multiple Documents – Updates all documents that meet a certain filter.

```

from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Define the filter and the update
filter_query = {"city": "Pune"}
new_values = {"$set": {"course": "M.Sc."}}

# Step 5: Perform the update
result = collection.update_many(filter_query, new_values)

# Step 6: Output result
print(f"{result.modified_count} document(s) updated.")

```

output

2 document(s) updated.

7.Delete a Single Document – Removes one document based on a condition.

```

from pymongo import MongoClient

```

```

# Step 1: Connect to MongoDB

```

```
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database
```

```
db = client["mydatabase"]
```

```
# Step 3: Select the collection
```

```
collection = db["students"]
```

```
# Step 4: Define the delete condition
```

```
delete_query = {"name": "Amit"}
```

```
# Step 5: Delete the first matching document
```

```
result = collection.delete_one(delete_query)
```

```
# Step 6: Output result
```

```
if result.deleted_count > 0:
```

```
    print("Document deleted successfully.")
```

```
else:
```

```
    print("No matching document found.")
```

output

Document deleted successfully.

8.Delete Multiple Documents – Deletes all documents that satisfy a condition.

```
from pymongo import MongoClient
```

```
# Step 1: Connect to MongoDB
```

```
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database
```

```
db = client["mydatabase"]
```

```
# Step 3: Select the collection
```

```
collection = db["students"]
```

```
# Step 4: Define the condition for deletion
```

```
delete_query = {"city": "Pune"}
```

```
# Step 5: Delete all matching documents
```

```
result = collection.delete_many(delete_query)
```

```
# Step 6: Output result
```

```
print(f"{result.deleted_count} document(s) deleted.")
```

output

2 document(s) deleted.

9.Find Documents with Projection – Retrieves documents but only returns specific fields.

```
from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Define projection (1 = include, 0 = exclude)
projection = {"_id": 0, "name": 1, "city": 1}

# Step 5: Find all documents with projection
results = collection.find({}, projection)

# Step 6: Print results
print("Documents (name and city only):")
for doc in results:
    print(doc)
```

output

```
Documents (name and city only):
{'name': 'Rahul', 'city': 'Mumbai'}
{'name': 'Priya', 'city': 'Chennai'}
```

10.Sort Documents – Finds and sorts documents based on one or more fields.

```
from pymongo import MongoClient
from pymongo import ASCENDING, DESCENDING # For sorting constants
```

```
# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Find and sort documents by age (ascending)
results = collection.find().sort("age", ASCENDING)

# Step 5: Print results
print("Students sorted by age (ascending):")
for doc in results:
    print(doc)
```

output

Students sorted by age (ascending):

```
{'_id': ..., 'name': 'Priya', 'age': 21, 'course': 'BBA', 'city': 'Chennai'}  
{'_id': ..., 'name': 'Rahul', 'age': 22, 'course': 'BCA', 'city': 'Mumbai'}  
{'_id': ..., 'name': 'Amit', 'age': 23, 'course': 'MBA', 'city': 'Delhi'}  
{'_id': ..., 'name': 'Sneha', 'age': 24, 'course': 'M.Sc.', 'city': 'Pune'}
```

11.Limit Query Results – Retrieves a limited number of documents from a query.

```
from pymongo import MongoClient
```

```
# Step 1: Connect to MongoDB
```

```
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database
```

```
db = client["mydatabase"]
```

```
# Step 3: Select the collection
```

```
collection = db["students"]
```

```
# Step 4: Retrieve only the first 3 documents
```

```
results = collection.find().limit(3)
```

```
# Step 5: Print results
```

```
print("First 3 documents:")
```

```
for doc in results:
```

```
    print(doc)
```

output

First 3 documents:

```
{'_id': ..., 'name': 'Rahul', 'age': 22, 'course': 'BCA', 'city': 'Mumbai'}  
{'_id': ..., 'name': 'Sneha', 'age': 24, 'course': 'MCA', 'city': 'Pune'}  
{'_id': ..., 'name': 'Amit', 'age': 23, 'course': 'MBA', 'city': 'Delhi'}
```

12.Skip Documents in Query – Skips a specified number of documents and returns the rest.

```
from pymongo import MongoClient
```

```
# Step 1: Connect to MongoDB
```

```
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database
```

```
db = client["mydatabase"]
```

```
# Step 3: Select the collection
```

```
collection = db["students"]
```

```
# Step 4: Skip the first 2 documents and retrieve the rest
results = collection.find().skip(2)
```

```
# Step 5: Print results
print("Documents after skipping the first 2:")
for doc in results:
    print(doc)
```

output

```
Documents after skipping the first 2:
{'_id': 1, 'name': 'Amit', 'age': 23, 'course': 'MBA', 'city': 'Delhi'}
{'_id': 2, 'name': 'Priya', 'age': 21, 'course': 'BBA', 'city': 'Chennai'}
```

13.Create an Index – Creates an index on one or more fields to speed up queries.

```
from pymongo import MongoClient
```

```
# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database
db = client["mydatabase"]
```

```
# Step 3: Select the collection
collection = db["students"]
```

```
# Step 4: Create an index on the 'name' field
index_name = collection.create_index([("name", 1)]) # 1 for ascending order
```

```
# Step 5: Output result
print(f"Index created: {index_name}")
```

output

```
Index created: name_1
```

14.Drop an Index – Removes an index from a collection.

```
from pymongo import MongoClient
```

```
# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database
db = client["mydatabase"]
```

```
# Step 3: Select the collection
collection = db["students"]
```

```
# Step 4: Drop the index
index_name = "name_1" # The name of the index to drop
```

```
collection.drop_index(index_name)
```

```
# Step 5: Output result  
print(f"Index '{index_name}' dropped successfully.")
```

output

```
Index 'name_1' dropped successfully.
```

15. Aggregate with \$group – Groups documents by a field and performs aggregations like sum or average.

```
from pymongo import MongoClient
```

```
# Step 1: Connect to MongoDB  
client = MongoClient("mongodb://localhost:27017/")
```

```
# Step 2: Select the database  
db = client["mydatabase"]
```

```
# Step 3: Select the collection  
collection = db["students"]
```

```
# Step 4: Perform aggregation with $group  
pipeline = [  
    {  
        "$group": {  
            "_id": "$city", # Group by 'city'  
            "average_age": {"$avg": "$age"}, # Calculate average age  
            "total_students": {"$sum": 1} # Count total students in each city  
        }  
    }  
]
```

```
# Step 5: Execute the aggregation pipeline  
results = collection.aggregate(pipeline)
```

```
# Step 6: Print results  
print("Aggregated Results (City, Average Age, Total Students):")  
for doc in results:  
    print(doc)
```

output

```
Aggregated Results (City, Average Age, Total Students):  
{'_id': 'Mumbai', 'average_age': 23.5, 'total_students': 2}  
{'_id': 'Chennai', 'average_age': 21.0, 'total_students': 1}  
{'_id': 'Pune', 'average_age': 24.0, 'total_students': 1}
```

16. Aggregate with \$match and \$project – Filters and reshapes documents in an aggregation pipeline.

```
from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Perform aggregation with $match and $project
pipeline = [
    {
        "$match": {
            "age": {"$gt": 22} # Filter students with age greater than 22
        }
    },
    {
        "$project": {
            "_id": 0, # Exclude the '_id' field
            "name": 1, # Include the 'name' field
            "age": 1, # Include the 'age' field
            "city": 1 # Include the 'city' field
        }
    }
]

# Step 5: Execute the aggregation pipeline
results = collection.aggregate(pipeline)

# Step 6: Print results
print("Filtered and reshaped results (age > 22):")
for doc in results:
    print(doc)
```

output

```
Filtered and reshaped results (age > 22):
{'name': 'Amit', 'age': 23, 'city': 'Delhi'}
{'name': 'Sneha', 'age': 24, 'city': 'Pune'}
```

17. Check if Collection Exists – Checks whether a collection exists in the database.

```
from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
```

```

db = client["mydatabase"]

# Step 3: Check if the collection exists
collection_name = "students"

if collection_name in db.list_collection_names():
    print(f"The collection '{collection_name}' exists.")
else:
    print(f"The collection '{collection_name}' does not exist.")

```

output

The collection 'students' exists.

18.Count Documents – Returns the count of documents that match a query.

```

from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Count documents where age > 22
count = collection.count_documents({"age": {"$gt": 22}})

# Step 5: Print the count
print(f"Number of students older than 22: {count}")

```

output

Number of students older than 22: 2

19.Paginate Results – Implements pagination to retrieve documents in chunks or pages.

```

from pymongo import MongoClient

# Step 1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Step 2: Select the database
db = client["mydatabase"]

# Step 3: Select the collection
collection = db["students"]

# Step 4: Define the number of results per page
results_per_page = 2

```

```
# Step 5: Get the page number (let's assume we are retrieving page 1)
page_number = 1
```

```
# Step 6: Calculate the number of documents to skip
skip = (page_number - 1) * results_per_page
```

```
# Step 7: Retrieve documents for the current page
results = collection.find().skip(skip).limit(results_per_page)
```

```
# Step 8: Print results
print(f"Results for page {page_number}:")
for doc in results:
    print(doc)
```

output

Results for page 1:

```
{'_id': ..., 'name': 'Rahul', 'age': 22, 'course': 'BCA', 'city': 'Mumbai'}
{'_id': ..., 'name': 'Sneha', 'age': 24, 'course': 'MCA', 'city': 'Pune'}
```

Results for page 2:

```
{'_id': ..., 'name': 'Amit', 'age': 23, 'course': 'MBA', 'city': 'Delhi'}
{'_id': ..., 'name': 'Priya', 'age': 21, 'course': 'BBA', 'city': 'Chennai'}
```

20. Connect to MongoDB Atlas – Connects to a remote MongoDB cluster hosted on MongoDB Atlas.

```
from pymongo import MongoClient
```

```
# Step 1: MongoDB Atlas connection string
atlas_connection_string = "mongodb+srv://<Yash>:<hari123>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority"
```

```
# Step 2: Connect to MongoDB Atlas
client = MongoClient(atlas_connection_string)
```

```
# Step 3: Select the database and collection
db = client["mydatabase"] # Replace 'mydatabase' with your database name
collection = db["students"] # Replace 'students' with your collection name
```

```
# Step 4: Perform a simple query to check the connection
result = collection.find_one() # Retrieve the first document from the collection
```

```
# Step 5: Print the result
if result:
    print("Connection successful! Here's a document:")
    print(result)
else:
```

```
print("No documents found.")
```

output

Connection successful! Here's a document:

```
{'_id': ObjectId('...'), 'name': 'Rahul', 'age': 22, 'course': 'BCA', 'city': 'Mumbai'}
```


Practical No 12

1. Create a package named library and implement few functions of library in python.

```
# library/__init__.py
from .books import add_book
from .search import search_book
from .display import display_books
```

books.py

```
# library/books.py
library_collection = []

def add_book(title, author):
    book = {'title': title, 'author': author}
    library_collection.append(book)
    print(f"Book added: {title} by {author}")
```

search.py

```
# library/search.py
from .books import library_collection

def search_book(title):
    for book in library_collection:
        if book['title'].lower() == title.lower():
            print(f"Book found: {book['title']} by {book['author']}")
            return
    print("Book not found.")
```

display.py

```
# library/display.py
from .books import library_collection

def display_books():
    if not library_collection:
        print("Library is empty.")
        return
    print("Library Collection:")
    for idx, book in enumerate(library_collection, 1):
        print(f"{idx}. {book['title']} by {book['author']}")
```

```

main.py
# main.py
from library import add_book, search_book, display_books

add_book("The Alchemist", "Paulo Coelho")
add_book("1984", "George Orwell")
add_book("Python Basics", "Yash Jawarkar")

search_book("1984")
display_books()

```

output :

```

Book added: The Alchemist by Paulo Coelho
Book added: 1984 by George Orwell
Book added: Python Basics by Yash Jawarkar
Book found: 1984 by George Orwell
Library Collection:
1. The Alchemist by Paulo Coelho
2. 1984 by George Orwell
3. Python Basics by Yash Jawarkar

```

2. Create a module in python to perform simple calculator operations

```

# calculator.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Error: Division by zero is not allowed."
    return a / b

# main.py
import calculator

a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

print("Select Operation:")

```

```
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

choice = input("Enter choice (1/2/3/4): ")

if choice == '1':
    print("Result:", calculator.add(a, b))
elif choice == '2':
    print("Result:", calculator.subtract(a, b))
elif choice == '3':
    print("Result:", calculator.multiply(a, b))
elif choice == '4':
    print("Result:", calculator.divide(a, b))
else:
    print("Invalid choice.")
```

Output

```
Enter first number: 10
Enter second number: 20
Select Operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1/2/3/4): 1
Result: 30.0
```

Practical No 13

Create a Python program that simulates a basic ATM interface. The program will involve a class named ATM with encapsulated attributes like `__balance` and `__pin`. Access to these private attributes will be managed through public methods, demonstrating the concept of data hiding and controlled access.

- Define a class with private data members.
- Use getter and setter methods to manipulate private data.
- Implement methods for balance check, deposit, and withdraw, with PIN verification.
- Use conditionals and input handling for user interaction.

```
class ATM:
```

```
    def __init__(self):
```

```
        self.__balance = 0.0
```

```
        self.__pin = None
```

```
    # Setter method for PIN (only once)
```

```
    def set_pin(self, pin):
```

```
        if self.__pin is None:
```

```
            self.__pin = pin
```

```
            print("PIN set successfully.")
```

```
        else:
```

```
            print("PIN already set. Cannot reset PIN in this version.")
```

```
    # PIN verification method
```

```
    def verify_pin(self):
```

```
        entered_pin = input("Enter your PIN: ")
```

```
        return entered_pin == self.__pin
```

```

#Getter method for balance
def get_balance(self):
    if self.verify_pin():
        print(f"Your current balance is: ₹{self.__balance:.2f}")
    else:
        print("Incorrect PIN!")

# Method to deposit money
def deposit(self):
    if self.verify_pin():
        try:
            amount = float(input("Enter amount to deposit: ₹"))
            if amount > 0:
                self.__balance += amount
                print(f"₹{amount:.2f} deposited successfully.")
            else:
                print("Invalid amount. Please enter a positive value.")
        except ValueError:
            print("Invalid input. Please enter a numeric value.")
    else:
        print("Incorrect PIN!")

# Method to withdraw money
def withdraw(self):
    if self.verify_pin():
        try:
            amount = float(input("Enter amount to withdraw: ₹"))
            if 0 < amount <= self.__balance:

```

```

        self.__balance -= amount

        print(f"₹{amount:.2f} withdrawn successfully.")
    else:
        print("Invalid amount or insufficient balance.")
    except ValueError:
        print("Invalid input. Please enter a numeric value.")
    else:
        print("Incorrect PIN!")

# ---- Main Program ----

def main():
    atm = ATM()

    print("Welcome to Secure ATM")
    pin = input("Set a 4-digit PIN to activate your ATM account: ")
    atm.set_pin(pin)

    while True:
        print("\nATM Menu:")
        print("1. Check Balance")
        print("2. Deposit Money")
        print("3. Withdraw Money")
        print("4. Exit")

        choice = input("Choose an option (1-4): ")

        if choice == '1':
            atm.get_balance()

```

```

elif choice == '2':
    atm.deposit()
elif choice == '3':
    atm.withdraw()
elif choice == '4':
    print("Thank you for using the ATM. Goodbye!")
    break
else:
    print("Invalid choice. Please try again.")

```

Run the program

```

if __name__ == "__main__":
    main()

```

Output

```

1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice: 1
Enter your PIN: 1234
Your balance is: ₹5000

--- ATM Menu ---
Enter your choice: 2
Enter amount to deposit: 1000
Enter your PIN: 1234
₹1000 deposited. New balance: ₹6000

--- ATM Menu ---
Enter your choice: 3
Enter amount to withdraw: 1500
Enter your PIN: 1234
₹1500 withdrawn. Remaining balance: ₹4500

```