JAGAN INSTITUTE OF MANAGEMENT STUDIES

SECTOR – 5, ROHINI,NEW DELHI

(Affiliated to)

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

SECTOR – 16 C, DWARKA, NEW DELHI

PRACTICAL FILE : Python

Submitted To : Dr. Deepti Sharma
Professor(IT)

Submitted By: **Shivansh**
Enrollment No.: 00214004424

MCA Ist Year (Section - A)[2nd Semester]

ACKNOWLEGEMENT

I take this opportunity to present my vote of thanks to my faculty who really acted as pillars to help my way throughout the execution of lab exercises that has led to successful and satisfactory completion of the study of Subject (Python).

I feel great sense of gratitude for **Dr. Deepti Sharma** under whose guidance and motivation this work has been performed.

I would also like to express my thanks to all **lab assistants** for giving me opportunity to work under their esteemed guidance. This project would not have completed without their guidance and coordination.

The inspiration of the faculty members of the Information Technology Department of JIMS Rohini (Sec-5) enabled me to make a thorough study of these subjects.

Student Name : Shivansh

Enrollment No. : 00214004424

CERTIFICATE

This is certified to be the bonafide work of the student,

Name: **Shivansh** , Enrollment No.: **00214004424** for the purpose of subject **Python** of MCA, 2nd semester under the supervision of **Dr. Deepti Sharma** during the academic year 2025-2026.

Dr. Deepti Sharma

Professor (IT)

JIMS, Rohini

1. Find the Second Largest Element in a List

```
def second_largest(num_list):
    if len(num_list) < 2:
        return "List needs at least 2 elements"
    unique_sorted = sorted(set(num_list), reverse=True)
    return unique_sorted[1] if len(unique_sorted) > 1 else "No second largest element found"

Num = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
print("Second largest element:", second_largest(Num))
```

2. Remove Odd and Negative Numbers from a List

```
def remove_odd_and_negative(num_list):
    return [num for num in num_list if num % 2 == 0 and num > 0]

numbers = [10, -5, 51, 2, -18, 4, -31, 13, 5, -23, 64, 29]
print("After removing odd and negative numbers:", remove_odd_and_negative(numbers))
```

3. Find Elements Occurring Odd Number of Times

```
def find_odd_occurrence(num_list):
    count_dict = {}

    for num in num_list:
        count_dict[num] = count_dict.get(num, 0) + 1

    return [num for num, count in count_dict.items() if count % 2 != 0]

numbers = [1, 2, 3, 2, 3, 1, 3, 4, 5, 4, 5, 5]
print("Elements occurring odd number of times:", find_odd_occurrence(numbers))
```

4. Check if a String is a Palindrome

```
def is_palindrome(s):
    s = s.replace(" ", "").lower()
    return s == s[::-1]

test_strings = ["radar", "Hello", "A man a plan a canal Panama", "Python"]
for string in test_strings:
    print(f"'{string}' is palindrome: {is_palindrome(string)}")
```

5. Check if a Substring is Present in a String

```
def is_substring_present(main_string, substring):
    return substring in main_string

main = "Python Programming is fun"
print(is_substring_present(main, "Programming")) # True
print(is_substring_present(main, "Java")) # False
```

6. Check if Parentheses are Balanced

```
def has_balanced_parentheses(string):
    stack = []

    brackets = {')': '(', '}': '{', ']': '['}

    for char in string:
        if char in brackets.values():
            stack.append(char)
        elif char in brackets.keys():
            if not stack or stack.pop() != brackets[char]:
                return False

    return not stack

print(has_balanced_parentheses("({[]})")) # True
```

7. Find Letters in First String but Not in Second

```
def letters_in_first_only(str1, str2):
    return ''.join(sorted(set(str1) - set(str2)))

print(letters_in_first_only("programming", "coding")) # 'ampr'
```

8. Capitalize Every Other Letter in a String

```
def capitalize_alternate(s):
    return ''.join(s[i].upper() if i % 2 else s[i].lower() for i in range(len(s)))

print(capitalize_alternate("corona")) # 'cOrOnA'
```

9. Remove a Key from a Dictionary

```
def remove_key(dictionary, key):
    return dictionary.pop(key, f"Key '{key}' not found")

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
print(remove_key(d, 'c')) # Key 'c' removed successfully
```

10. Count Word Frequency in a String

```
def count_word_frequency(text):
    for char in '.,;:!?"()[]{}':
        text = text.replace(char, '')
    words = text.lower().split()
    return {word: words.count(word) for word in set(words)}

text = "Python is a programming language. Python is fun."
print(count_word_frequency(text))
```

11. Store and Display Student Information

```
def store_student_info():
    students = {}

    n = int(input("Enter number of students: "))
    for _ in range(n):
        adm_num = input("Admission Number: ")
        students[adm_num] = {
            'roll_number': input("Roll Number: "),
            'name': input("Name: "),
            'marks': float(input("Marks: "))
        }

    return students
```

12. Find Sum of a Nested List Using Recursion

```
def calculate_sum(nested_list):
    return sum(calculate_sum(e) if isinstance(e, list) else e for e in nested_list)

nested_list = [1, 2, [3, 4], [5, [6, 7]], 8, [9, 10]]
print(calculate_sum(nested_list)) # 55
```

13. Append a String to a File

```
def append_to_file(filename, text):
    with open(filename, 'a') as file:
        file.write(text + '\n')

append_to_file("sample.txt", "This is a sample text")
```

14. Count Occurrences of a Word in a File

```
def count_word_in_file(filename, word):
    try:
        with open(filename, 'r') as file:
            content = file.read().lower()

        return content.split().count(word.lower())
    except FileNotFoundError:
        return "File not found"
```

15. Compare Two Files and Display Line Differences

```
def compare_files(file1, file2):
    with open(file1, 'r') as f1, open(file2, 'r') as f2:
        lines1, lines2 = f1.readlines(), f2.readlines()

    return f"File1: {len(lines1)} lines, File2: {len(lines2)} lines"
```

16. Find Index of a Key Using Linear Search

```
def linear_search(arr, key):
    return arr.index(key) if key in arr else -1

print(linear_search([10, 51, 2, 18], 18)) # 3
```

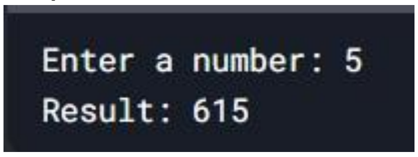
17. Sort a List Using Bubble Sort

```
def bubble_sort(arr):
    for i in range(len(arr)):
        for j in range(len(arr)-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr

print(bubble_sort([10, 51, 2, 18])) # [2, 10, 18, 51]
```

18. Read a number n and compute n + nn + nnn

```
n = input("Enter a number: ")
nn = n + n
nnn = n + n + n
result = int(n) + int(nn) + int(nnn)
print(f"Result: {result}")
```

Output:

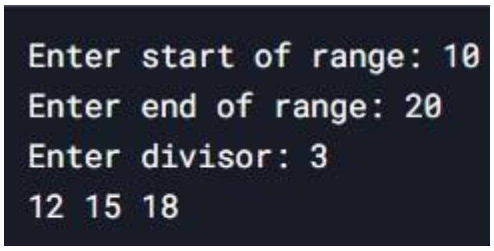


```
Enter a number: 5
Result: 615
```

19. Print all Numbers in a Range Divisible by a Given Number

```
start = int(input("Enter start of range: "))
end = int(input("Enter end of range: "))
divisor = int(input("Enter divisor: "))
for num in range(start, end + 1):
    if num % divisor == 0:
        print(num, end=" ")
```

Output:

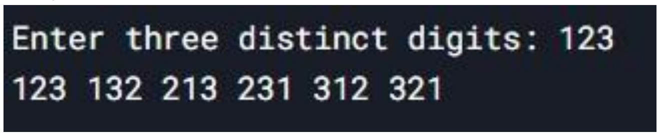


```
Enter start of range: 10
Enter end of range: 20
Enter divisor: 3
12 15 18
```

20. Accept three distinct digits and print all possible combinations

```
from itertools import permutations
digits = input("Enter three distinct digits: ")
combinations = permutations(digits, 3)
for combo in combinations:
    print("".join(combo), end=" ")
```

Output:

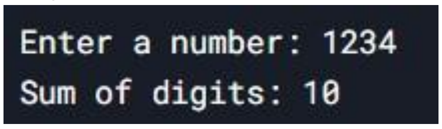


```
Enter three distinct digits: 123
123 132 213 231 312 321
```

21. Program to Find the Sum of Digits in a Number

```
num = input("Enter a number: ")
sum_digits = sum(int(digit) for digit in num)
print(f"Sum of digits: {sum_digits}")
```

Output:

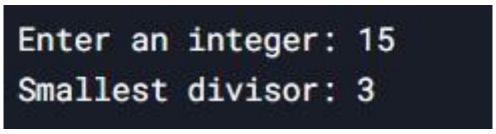


```
Enter a number: 1234
Sum of digits: 10
```

22. Program to Find the Smallest Divisor of an Integer (other than 1)

```
num = int(input("Enter an integer: "))
for i in range(2, num + 1):
    if num % i == 0:
        print(f"Smallest divisor: {i}")
        break
```

Output:

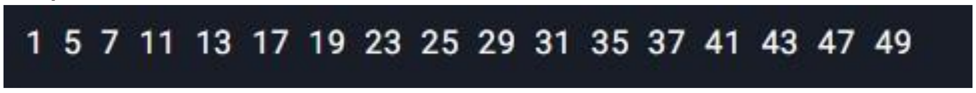


```
Enter an integer: 15
Smallest divisor: 3
```

23. Print all integers that are not divisible by either 2 or 3 and lie between 1 and 50

```
for num in range(1, 51):
    if num % 2 != 0 and num % 3 != 0:
        print(num, end=" ")
```

Output:

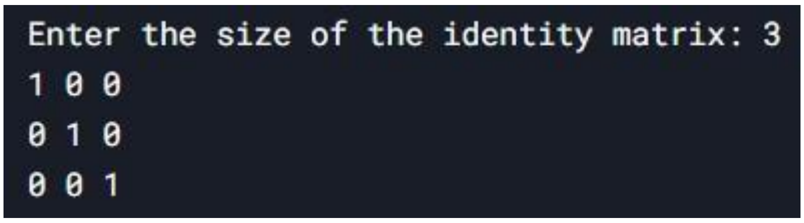


```
1 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49
```

24. Accept a number n and print an identity matrix of N*N

```
n = int(input("Enter the size of the identity matrix: "))
for i in range(n):
    for j in range(n):
        print(1 if i == j else 0, end=" ")
    print()
```

Output:

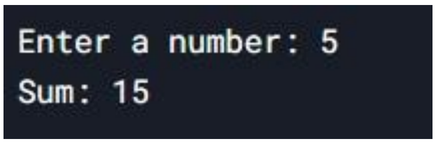


```
Enter the size of the identity matrix: 3
1 0 0
0 1 0
0 0 1
```

25. Compute the Value of 1 + 2 + 3 + ... + n

```
n = int(input("Enter a number: "))
result = n * (n + 1) // 2
print(f"Sum: {result}")
```

Output:

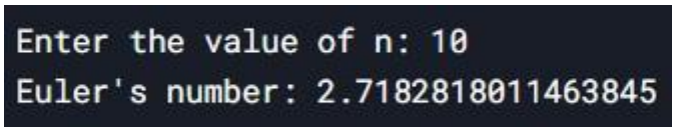


```
Enter a number: 5
Sum: 15
```

26. Compute Euler's Number (e)

```
import math
n = int(input("Enter the value of n: "))
e = sum(1 / math.factorial(i) for i in range(n + 1))
print(f"Euler's number: {e}")
```

Output:

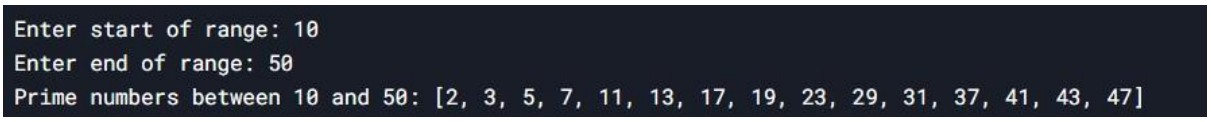


```
Enter the value of n: 10
Euler's number: 2.7182818011463845
```

27. Print Prime Numbers in a Range using Sieve of Eratosthenes

```
def sieve_of_eratosthenes(limit):
    primes = [True] * (limit + 1)
    primes[0] = primes[1] = False
    for num in range(2, int(limit**0.5) + 1):
        if primes[num]:
            for multiple in range(num * num, limit + 1, num):
                primes[multiple] = False
    return [num for num, is_prime in enumerate(primes) if is_prime]

start = int(input("Enter start of range: "))
end = int(input("Enter end of range: "))
print(f"Prime numbers between {start} and {end}: {sieve_of_eratosthenes(end)}")
```

Output:

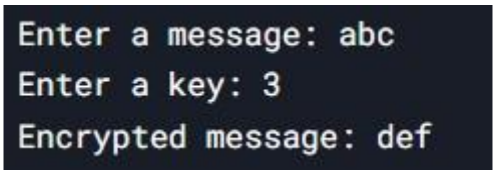


```
Enter start of range: 10
Enter end of range: 50
Prime numbers between 10 and 50: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

28. Write a program that encrypts a message by adding a key value to every character. (Caesar Cipher)

```
message = input("Enter a message: ")
key = int(input("Enter a key: "))
encrypted = "".join(chr(ord(char) + key) for char in message)
print(f"Encrypted message: {encrypted}")
```

Output:



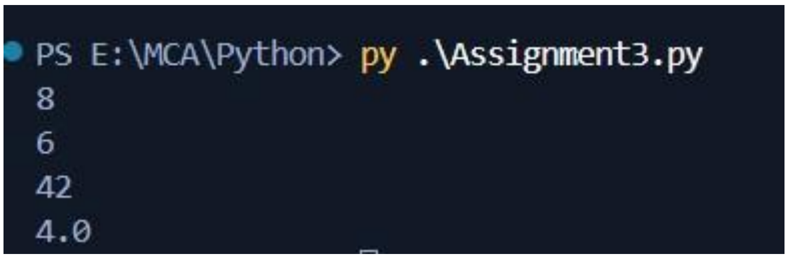
```
Enter a message: abc
Enter a key: 3
Encrypted message: def
```

Assignment - III

1. Create a Class which Performs Basic Calculator Operations

```
class Calculator:
    def add(self, a, b):
        return a + b

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

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

    def divide(self, a, b):
        if b == 0:
            return "Error! Division by zero."
        return a / b

calc = Calculator()
print(calc.add(5, 3))
print(calc.subtract(10, 4))
print(calc.multiply(6, 7))
print(calc.divide(8, 2))
```

Output:

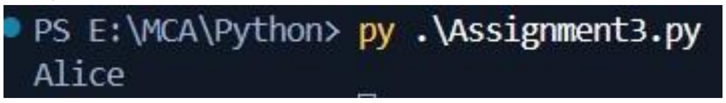


```
PS E:\MCA\Python> py .\Assignment3.py
8
6
42
4.0
```

2. Voting System

```
from collections import Counter

class VotingSystem:
    def __init__(self):
        self.votes = []

    def cast_vote(self, candidate_name):
        self.votes.append(candidate_name)

    def get_winner(self):
        if not self.votes:
            return "No votes cast"
        vote_count = Counter(self.votes)
        max_votes = max(vote_count.values())

        winners = sorted([name for name, count in vote_count.items() if count == max_votes])

        return winners[0]

# Example Usage
election = VotingSystem()
election.cast_vote("Alice")
election.cast_vote("Bob")
election.cast_vote("Alice")
election.cast_vote("Charlie")
election.cast_vote("Bob")

print(election.get_winner()) # Should print the lexicographically smallest winner
```

Output:

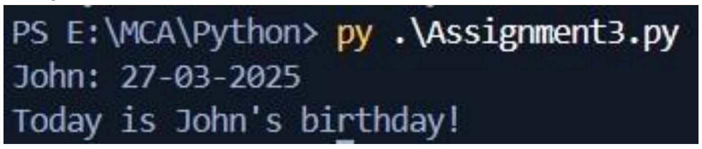


```
PS E:\MCA\Python> py .\Assignment3.py
Alice
```

3. Create Birthday Reminder Application

```
import datetime

class BirthdayReminder:
    def __init__(self, filename="birthdays.txt"):
        self.filename = filename

    def add_birthday(self, name, date_of_birth):
        try:
            with open(self.filename, "a") as file:
                file.write(f"{name},{date_of_birth}\n")
        except Exception as e:
            print("Error writing to file:", e)

    def show_birthdays(self):
        try:
            with open(self.filename, "r") as file:
                for line in file:
                    name, dob = line.strip().split(",")
                    print(f"{name}: {dob}")
        except FileNotFoundError:
            print("No birthdays found.")

    def check_today_birthdays(self):
        today = datetime.datetime.today().strftime("%d-%m")
        try:
            with open(self.filename, "r") as file:
                for line in file:
                    name, dob = line.strip().split(",")
                    if today in dob:
                        print(f"Today is {name}'s birthday!")
        except FileNotFoundError:
            print("No birthdays found.")

# Example Usage
reminder = BirthdayReminder()
reminder.add_birthday("John", "27-03-1995")
reminder.show_birthdays()
reminder.check_today_birthdays()
```

Output:



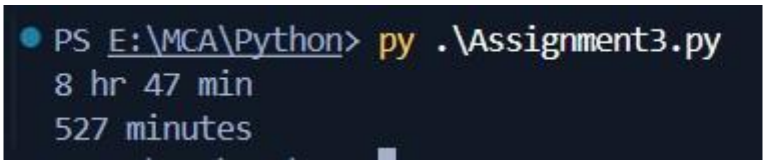
```
PS E:\MCA\Python> py .\Assignment3.py
John: 27-03-2025
Today is John's birthday!
```

4. Create a class "Time"

```
class Time:
    def __init__(self, hrs, mins):
        self.hrs = hrs
        self.mins = mins

    def add_time(self, other):
        total_mins = self.mins + other.mins
        total_hrs = self.hrs + other.hrs + total_mins // 60
        total_mins %= 60

        return Time(total_hrs, total_mins)

    def show_time(self):
        print(f"{self.hrs} hr {self.mins} min")

    def show_minute(self):
        print(self.hrs * 60 + self.mins, "minutes")
```

```
# Example Usage
t1 = Time(6, 35)
t2 = Time(2, 12)
t3 = t1.add_time(t2)
t3.show_time()
t3.show_minute()
```

Output:

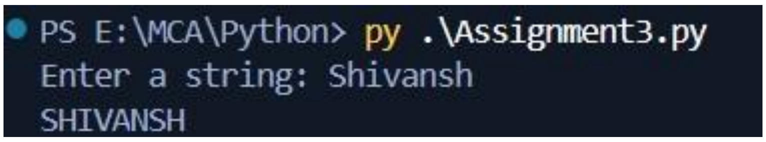


```
PS E:\MCA\Python> py .\Assignment3.py
8 hr 47 min
527 minutes
```

5. Write a Python class for string manipulation

```
class StringManipulation:
    def get_String(self):
        self.text = input("Enter a string: ")

    def print_String(self):
        print(self.text.upper())

# Example Usage
s = StringManipulation()
s.get_String()
s.print_String()
```

Output:

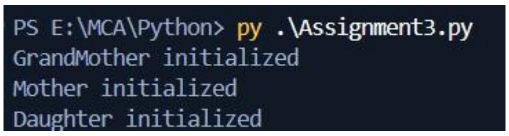


```
PS E:\MCA\Python> py .\Assignment3.py
Enter a string: Shivansh
SHIVANSH
```

6. Create class inheritance hierarchy

```
class GrandMother:
    def __init__(self):
        print("GrandMother initialized")

class Mother(GrandMother):
    def __init__(self):
        super().__init__()
        print("Mother initialized")

class Daughter(Mother):
    def __init__(self):
        super().__init__()
        print("Daughter initialized")

# Example Usage
d = Daughter()
```

Output:

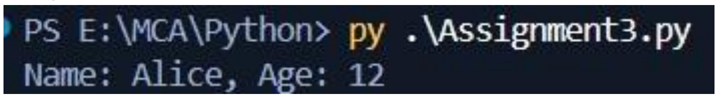


```
PS E:\MCA\Python> py .\Assignment3.py
GrandMother initialized
Mother initialized
Daughter initialized
```

7. Create a Parent-Child class relationship

```
class Parent:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def print_details(self):
        print(f"Name: {self.name}, Age: {self.age}")

class Child(Parent):
    def __init__(self, name, age):
        super().__init__(name, age)
        self.print_details()

# Example Usage
c = Child("Alice", 12)
```

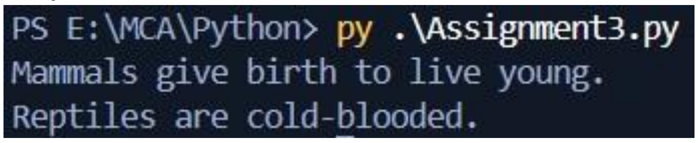
Output:

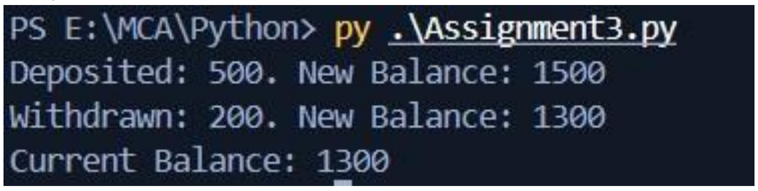


```
PS E:\MCA\Python> py .\Assignment3.py
Name: Alice, Age: 12
```

8. Create an abstract class with inherited subclasses

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def characteristic(self):
        pass

class Mammals(Animal):
    def characteristic(self):
        print("Mammals give birth to live young.")

class Reptiles(Animal):
    def characteristic(self):
        print("Reptiles are cold-blooded.")

class Birds(Animal):
    def characteristic(self):
        print("Birds have feathers and lay eggs.")

class Amphibians(Animal):
    def characteristic(self):
        print("Amphibians live both in water and on land.")
```

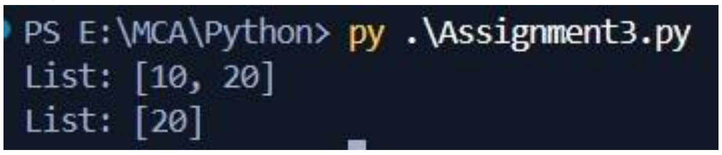
```
# Example Usage
m = Mammals()
m.characteristic()
r = Reptiles()
r.characteristic()
```
Output:

```
PS E:\MCA\Python> py .\Assignment3.py
Mammals give birth to live young.
Reptiles are cold-blooded.
```
9. Implement ATM simulation system

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

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: {amount}. New Balance: {self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds!")
        else:
            self.balance -= amount
            print(f"Withdrawn: {amount}. New Balance: {self.balance}")

    def check_balance(self):
        print(f"Current Balance: {self.balance}")

# Example Usage
atm = ATM(1000)
atm.deposit(500)
atm.withdraw(200)
atm.check_balance()
```
Output:

```
PS E:\MCA\Python> py .\Assignment3.py
Deposited: 500. New Balance: 1500
Withdrawn: 200. New Balance: 1300
Current Balance: 1300
```

10. Python Program to Append, Delete and Display Elements of a List Using Classes

```
class ListOperations:
    def __init__(self):
        self.lst = []

    def append_element(self, element):
        self.lst.append(element)
    def delete_element(self, element):
        if element in self.lst:
            self.lst.remove(element)
        else:
            print("Element not found!")

    def display_list(self):
        print("List:", self.lst)

# Example Usage
l = ListOperations()
l.append_element(10)
l.append_element(20)
l.display_list()
l.delete_element(10)
l.display_list()
```

Output:



```
PS E:\MCA\Python> py .\Assignment3.py
List: [10, 20]
List: [20]
```

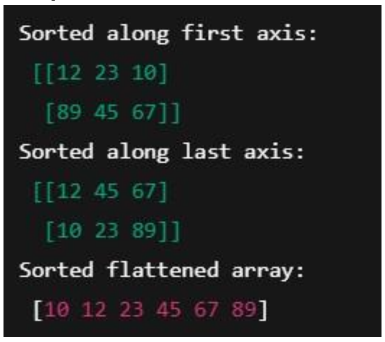
Assignment - IV

1. Sort a NumPy array along different axes

python

```
import numpy as np

arr = np.array([[12, 45, 67], [89, 23, 10]])

# Sort along the first axis (column-wise)
sorted_first_axis = np.sort(arr, axis=0)
print("Sorted along first axis:\n", sorted_first_axis)

# Sort along the last axis (row-wise)
sorted_last_axis = np.sort(arr, axis=1)
print("Sorted along last axis:\n", sorted_last_axis)

# Flattened sort
sorted_flattened = np.sort(arr, axis=None)
print("Sorted flattened array:\n", sorted_flattened)
```

Output:

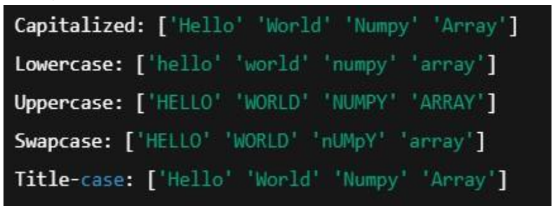


```
Sorted along first axis:
 [[12 23 10]
  [89 45 67]]
Sorted along last axis:
 [[12 45 67]
  [10 23 89]]
Sorted flattened array:
 [10 12 23 45 67 89]
```

2. String manipulations with NumPy arrays

```
import numpy as np

arr = np.array(['hello', 'world', 'NumPy', 'ARRAY'])

capitalized = np.char.capitalize(arr)
lowercase = np.char.lower(arr)
uppercase = np.char.upper(arr)
swapcase = np.char.swapcase(arr)
title_case = np.char.title(arr)

print("Capitalized:", capitalized)
print("Lowercase:", lowercase)
print("Uppercase:", uppercase)
print("Swapcase:", swapcase)
print("Title-case:", title_case)
```

Output:

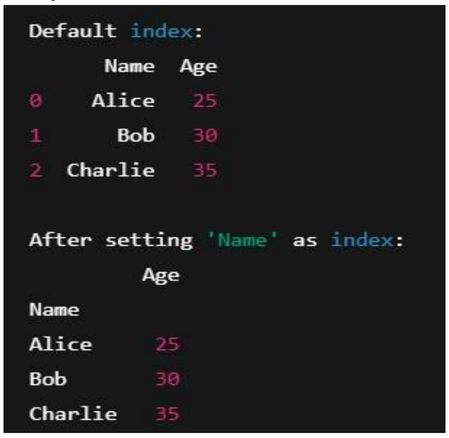


```
Capitalized: ['Hello' 'World' 'Numpy' 'Array']
Lowercase: ['hello' 'world' 'numpy' 'array']
Uppercase: ['HELLO' 'WORLD' 'NUMPY' 'ARRAY']
Swapcase: ['HELLO' 'WORLD' 'nUMpY' 'array']
Title-case: ['Hello' 'World' 'Numpy' 'Array']
```

3. Display default index and set column as index in Pandas

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}

df = pd.DataFrame(data)

print("Default index:")
print(df)

df_indexed = df.set_index('Name')
print("\nAfter setting 'Name' as index:")
print(df_indexed)
```

Output:

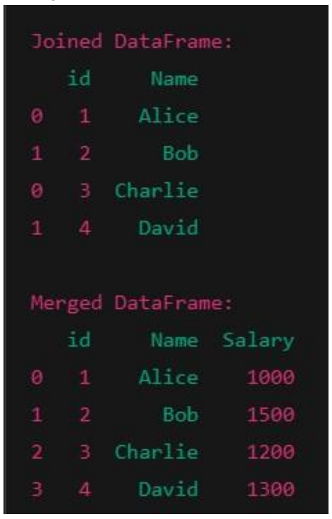


```
Default index:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35

After setting 'Name' as index:
         Age
Name
Alice     25
Bob       30
Charlie   35
```

4. Join and merge DataFrames in Pandas

```
df1 = pd.DataFrame({'id': [1, 2], 'Name': ['Alice', 'Bob']})
df2 = pd.DataFrame({'id': [3, 4], 'Name': ['Charlie', 'David']})
df3 = pd.DataFrame({'id': [1, 2, 3, 4], 'Salary': [1000, 1500, 1200, 1300]})

joined = pd.concat([df1, df2])
print("Joined DataFrame:\n", joined)

merged = pd.merge(joined, df3, on='id')
print("Merged DataFrame:\n", merged)
```

Output:

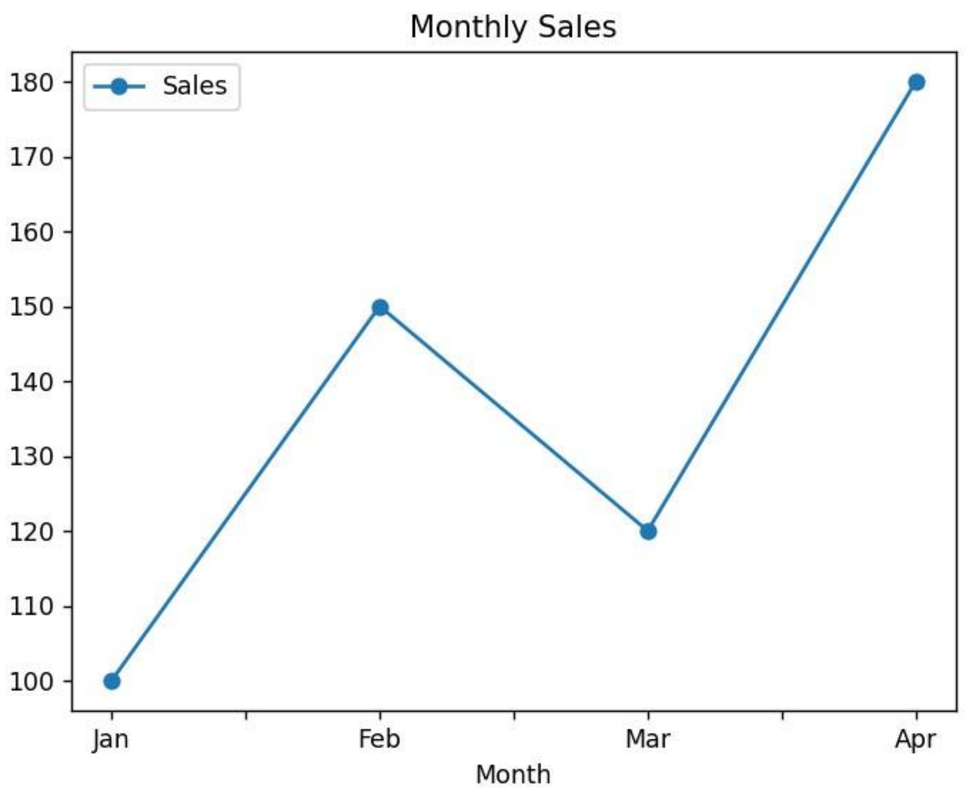


```
Joined DataFrame:
   id     Name
0   1    Alice
1   2      Bob
0   3  Charlie
1   4    David

Merged DataFrame:
   id     Name  Salary
0   1    Alice    1000
1   2      Bob    1500
2   3  Charlie    1200
3   4    David    1300
```

5. Analyze and visualize sales data

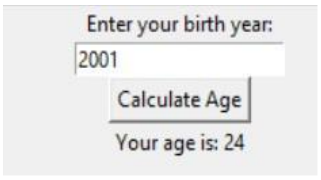
```
import pandas as pd
import matplotlib.pyplot as plt

sales_data = {'Month': ['Jan', 'Feb', 'Mar', 'Apr'],
        'Sales': [100, 150, 120, 180]}

df_sales = pd.DataFrame(sales_data)

df_sales.plot(kind='line', x='Month', y='Sales', marker='o', title='Monthly Sales')
plt.show()

df_sales.plot(kind='bar', x='Month', y='Sales', title='Monthly Sales (Bar)')
plt.show()

df_sales.set_index('Month')['Sales'].plot.pie(autopct='%1.1f%%', title='Sales Distribution')
plt.ylabel('')
plt.show()
```

Output:



6. Age calculator application using GUI widgets

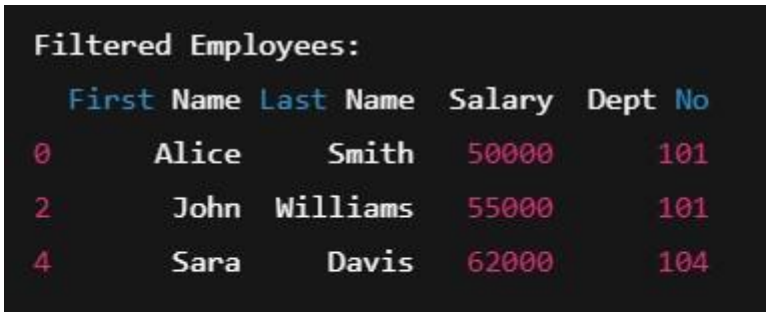
```
import tkinter as tk
from datetime import datetime

def calculate_age():
    birth_year = int(entry.get())
    current_year = datetime.now().year
    age = current_year - birth_year
    result_label.config(text=f"Your age is: {age}")

root = tk.Tk()
root.title("Age Calculator")

tk.Label(root, text="Enter your birth year:").pack()
entry = tk.Entry(root)
entry.pack()

tk.Button(root, text="Calculate Age", command=calculate_age).pack()
result_label = tk.Label(root, text="")
result_label.pack()

root.mainloop()
```
Output:



7. Filter employee data using Pandas

```
import pandas as pd

data = {
   'First Name': ['Alice', 'Mark', 'John', 'Michael', 'Sara'],
   'Last Name': ['Smith', 'Johnson', 'Williams', 'Brown', 'Davis'],
   'Salary': [50000, 60000, 55000, 58000, 62000],
   'Dept No': [101, 102, 101, 103, 104]
}

df = pd.DataFrame(data)

filtered = df[~df['First Name'].str.contains('m', case=False)]

print("Filtered Employees:\n", filtered[['First Name', 'Last Name', 'Salary', 'Dept No']])
```

Output:



```
Filtered Employees:
  First Name Last Name  Salary  Dept No
0      Alice     Smith   50000      101
2       John  Williams   55000      101
4       Sara     Davis   62000      104
```