1. Write a Python Program
- To check if a number is Positive, Negative or Zero
- To find Factorial of a Number
- To Print the Fibonacci sequence
- To Print Multiplication Table
- Python Program to Check Armstrong Number (A number is called Armstrong number if it is equal to the sum of the cubes of its own digits. For example: 153 is an Armstrong number since 153 = 1*1*1 + 5*5*5 + 3*3*3.)
Use appropriate conditional constructs, looping constructs and function.

Ans :

```
def check_number():
    num = float(input("Enter a number: "))
    if num > 0: print("\nOutput : Positive number")
    elif num == 0: print("\nOutput : Zero")
    else: print("\nOutput : Negative number")
def factorial():
    num = int(input("Enter a number to find its factorial: "))
    factorial = 1
    if num < 0: print("\nOutput : Factorial does not exist for negative numbers")
    elif num == 0: print("\nOutput : Factorial of 0 is 1")
    else:
        for i in range(1, num + 1):
            factorial *= i
        print("\nOutput : Factorial of", num, "is", factorial)
def fibonacci():
    n = int(input("Enter the number of terms: "))
    a, b = 0, 1
    count = 0
    if n <= 0: print("Please enter a positive integer")
    elif n == 1:
        print("\nOutput : Fibonacci sequence up to", n, ":")
        print(a)
    else:
        print("\nOutput : Fibonacci sequence:")
        while count < n:
            print(a, end=" ")
            nth = a + b
            a = b
            b = nth
            count += 1
def multiplication_table():
    num = int(input("Enter the number for which you want to print the multiplication table: "))
    print("\nOutput : Multiplication table of", num)
    for i in range(1, 11):
        print("", num, "x", i, "=", num * i)
def is_armstrong_number(num):
    order = len(str(num))
    sum = 0
```

```
    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** order
        temp //= 10
    return num == sum
def check_armstrong_number():
    num = int(input("Enter a number to check if it's an Armstrong number: "))
    if is_armstrong_number(num):
        print("\nOutput : ", num, "is an Armstrong number")
    else:
        print("\nOutput : ", num, "is not an Armstrong number")
# Main program
while True:
    print("\nChoose an option:")
    print("1. Check if a number is Positive, Negative or Zero")
    print("2. Find Factorial of a Number")
    print("3. Print the Fibonacci sequence")
    print("4. Print Multiplication Table")
    print("5. Check Armstrong Number")
    print("6. Exit")
    choice = input("Enter your choice (1-6): ")
    if choice == '1': check_number()
    elif choice == '2': factorial()
    elif choice == '3':fibonacci()
    elif choice == '4':multiplication_table()
    elif choice == '5':check_armstrong_number()
    elif choice == '6':
        print("Exiting the program...")
        break
    else: print("Invalid choice. Please enter a number between 1 and 6.")
```

Output

```
londhe@developer:~/Downloads/MCA/SEM-2/lab journals/python-programs-source-code$ python3 Program-1.py

Choose an option:
1. Check if a number is Positive, Negative or Zero
2. Find Factorial of a Number
3. Print the Fibonacci sequence
4. Print Multiplication Table
5. Check Armstrong Number
6. Exit
Enter your choice (1-6): 1
Enter a number: 1

Output : Positive number

Choose an option:
1. Check if a number is Positive, Negative or Zero
2. Find Factorial of a Number
3. Print the Fibonacci sequence
4. Print Multiplication Table
5. Check Armstrong Number
6. Exit
Enter your choice (1-6):
```

2. WAP to convert temperature of 10 cities into Fahrenheit using lambda and map function.

Ans :

```
"""
    Use following API to get current weather
     url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&units=metric"
"""
celsius_to_fahrenheit = lambda celsius: (celsius * 9/5) + 32

# List of temperatures in Celsius for 10 cities
temperatures_celsius = [22, 18, 25, 30, 16, 20, 27, 23, 19, 21]

# Using map function to apply the conversion function to each temperature
temperatures_fahrenheit = list(map(celsius_to_fahrenheit, temperatures_celsius))

# Displaying the temperatures in Fahrenheit for each city
for city, temperature in zip(range(1, 11), temperatures_fahrenheit):
    print(f"City {city}: {temperature:.2f}°F")
```

Output :

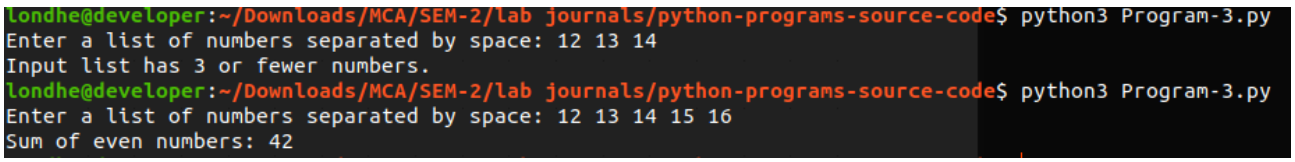
```
londhe@developer:~/Downloads/MCA/SEM-2/lab journals/python-programs-source-code$ python3 Program-2.py
City 1: 71.60°F
City 2: 64.40°F
City 3: 77.00°F
City 4: 86.00°F
City 5: 60.80°F
City 6: 68.00°F
City 7: 80.60°F
City 8: 73.40°F
City 9: 66.20°F
City 10: 69.80°F
```

3. WAP to accept a list of numbers as an input. If input is greater than 3 then perform addition of even numbers. Make a use of map, reduce and lambda function.

Ans :

```
from functools import reduce

is_even = lambda x: x % 2 == 0

addition = lambda x, y: x + y

def add_even_numbers(numbers):
    even_numbers = filter(is_even, numbers)
    result = reduce(addition, even_numbers)
    return result

# Main function
def main():
    try:
        numbers = list(map(int, input("Enter a list of numbers separated by space: ").split()))

        if len(numbers) > 3:
            result = add_even_numbers(numbers)
            print("Sum of even numbers:", result)
        else:
            print("Input list has 3 or fewer numbers.")
    except ValueError:
        print("Please enter valid numbers.")

if __name__ == "__main__":
    main()
```

Output :



4. WAP to create your own arithmetic module and perform arithmetic operations.

Ans :

arithmatic_module.py

```
def add(a, b):
    """Function to perform addition."""
    return a + b
def subtract(a, b):
    """Function to perform subtraction."""
    return a - b
def multiply(a, b):
    """Function to perform multiplication."""
    return a * b
def divide(a, b):
    """Function to perform division."""
    if b == 0:
        return "Error: Division by zero!"
    else:
        return a / b
```

Program-4.py

```
from helpers import arithmetic_modules

# Perform arithmetic operations
result_addition = arithmetic_modules.add(5, 3)
result_subtraction = arithmetic_modules.subtract(10, 4)
result_multiplication = arithmetic_modules.multiply(7, 2)
result_division = arithmetic_modules.divide(15, 3)

# Display results
print("Addition:", result_addition)
print("Subtraction:", result_subtraction)
print("Multiplication:", result_multiplication)
print("Division:", result_division)
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-4.py
Addition: 8
Subtraction: 6
Multiplication: 14
Division: 5.0
```

5. WAP to create Operator package and perform logical operations.

Ans :

logical_operations.py

```
def logical_and(a, b):
    """Perform logical AND operation."""
    return a and b

def logical_or(a, b):
    """Perform logical OR operation."""
    return a or b

def logical_not(a):
    """Perform logical NOT operation."""
    return not a
```

Program-5.py

```
from helpers import logical_operations

# Perform logical operations
result_and = logical_operations.logical_and(True, False)
result_or = logical_operations.logical_or(True, False)
result_not = logical_operations.logical_not(True)

# Display results
print("""Perform logical AND operation.""")
print("Logical AND:", result_and, "\n")
print("""Perform logical OR operation.""")
print("Logical OR:", result_or, "\n")
print("""Perform logical NOT operation.""")
print("Logical NOT:", result_not, "\n")
```

Output :

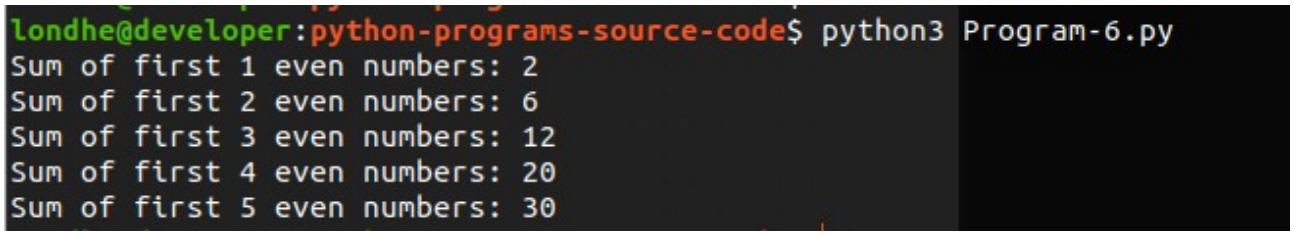
```
londhe@developer:python-programs-source-code$ python3 Program-5.py
Perform logical AND operation.
Logical AND: False

Perform logical OR operation.
Logical OR: True

Perform logical NOT operation.
Logical NOT: False
```

6. Write a generator function to perform sum of n even numbers.

Ans :

```
def sum_of_even_numbers(n):
    """Generator function to generate the sum of n even numbers."""
    count = 0
    total_sum = 0
    current_number = 0

    while count < n:
        current_number += 2  # Increment by 2 to get the next even number
        total_sum += current_number
        count += 1
        yield total_sum

# Example usage:
n = 5
sum_generator = sum_of_even_numbers(n)

for i, result in enumerate(sum_generator, start=1):
    print(f"Sum of first {i} even numbers:", result)
```

Output :



7. WAP to implement multiple decorators or chaining decorators.

Ans :

```
# Decorator function to convert a string to uppercase
def uppercase_decorator(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        return result.upper()
    return wrapper

# Decorator function to add a prefix to a string
def prefix_decorator(prefix):
    def decorator(func):
        def wrapper(*args, **kwargs):
            result = func(*args, **kwargs)
            return prefix + result
        return wrapper
    return decorator

# Decorator function to add a suffix to a string
def suffix_decorator(suffix):
    def decorator(func):
        def wrapper(*args, **kwargs):
            result = func(*args, **kwargs)
            return result + suffix
        return wrapper
    return decorator

# Function with multiple decorators chained
@suffix_decorator("!!!")
@prefix_decorator("Result: ")
@uppercase_decorator
def greet(name):
    return f"Hello, {name}"

# Using the decorated function
print(greet("Alice"))
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-7.py
Result: HELLO, ALICE!!!
londhe@developer:python-programs-source-code$
```

8. WAP to create abstract class and display the details. (Any suitable example)

Ans :

```
from abc import ABC, abstractmethod

# Define an abstract class
class Vehicle(ABC):
    def __init__(self, make, model):
        self.make = make
        self.model = model

    @abstractmethod
    def display_details(self):
        pass

# Define a subclass of Vehicle
class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model)
        self.year = year

    def display_details(self):
        print(f"Car Details:\nMake: {self.make}\nModel: {self.model}\nYear: {self.year}")

# Define another subclass of Vehicle
class Truck(Vehicle):
    def __init__(self, make, model, capacity):
        super().__init__(make, model)
        self.capacity = capacity
    def display_details(self):
        print(f"Truck Details:\nMake: {self.make}\nModel: {self.model}\nCapacity: {self.capacity}
tons")

# Create instances of subclasses and display details
car = Car("Toyota", "Camry", 2022)
truck = Truck("Ford", "F-150", 5)
car.display_details()
print()
truck.display_details()
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-8.py
Car Details:
Make: Toyota
Model: Camry
Year: 2022

Truck Details:
Make: Ford
Model: F-150
Capacity: 5 tons
```

9. WAP to implement encapsulation. (Any suitable example)

Ans :

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number  # Encapsulated attribute
        self.__balance = balance  # Encapsulated attribute
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited ${amount}. New balance: ${self.__balance}")
        else:
            print("Invalid deposit amount.")
    def withdraw(self, amount):
        """Method to withdraw money from the account."""
        if amount > 0 and amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrew ${amount}. New balance: ${self.__balance}")
        else:
            print("Insufficient funds or invalid withdrawal amount.")
    def get_balance(self):
        """Method to get the current balance."""
        return self.__balance
    def get_account_number(self):
        """Method to get the account number."""
        return self.__account_number
# Creating a BankAccount object
account = BankAccount("123456789", 1000)

# Accessing encapsulated attributes directly (shouldn't be done)
# print(account.__account_number)  # This will raise an AttributeError
# Accessing encapsulated attributes using getter methods
print("Account Number:", account.get_account_number())
print("Balance:", account.get_balance())
# Depositing and withdrawing money
account.deposit(500)
account.withdraw(200)
```

Output

```
londhe@developer:python-programs-source-code$ python3 Program-9.py
Traceback (most recent call last):
  File "Program-9.py", line 34, in <module>
    print(account.__account_number)  # This will raise an AttributeError
AttributeError: 'BankAccount' object has no attribute '__account_number'
londhe@developer:python-programs-source-code$ python3 Program-9.py
Account Number: 123456789
Balance: 1000
Deposited $500. New balance: $1500
Withdrew $200. New balance: $1300
```

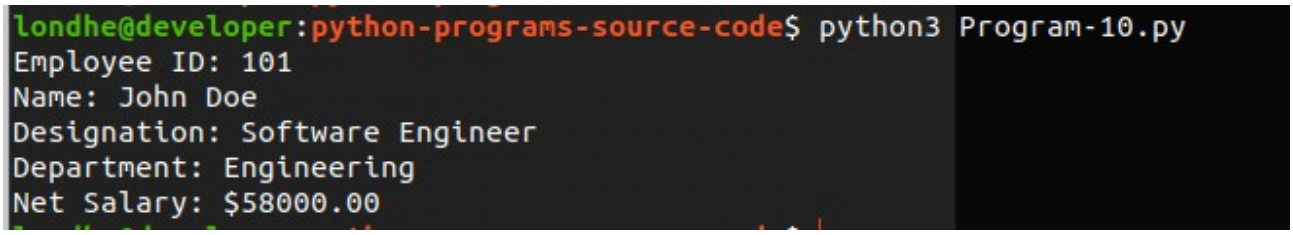
10. Python Program to Calculate the employee salary using Inheritance as Class A will have detail of the employee such as employee ID, Name, Designation, and Department. Class B will have Gross salary such as Basic Salary, DA, HRA,TA and Gross Salary. Class C will have Deduction Salary as Income tax and any other deduction. Class D will have Net Salary.

Ans :

```
class Employee:
    def __init__(self, emp_id, name, designation, department):
        self.emp_id = emp_id
        self.name = name
        self.designation = designation
        self.department = department

class GrossSalary(Employee):
    def __init__(self, emp_id, name, designation, department, basic_salary, da, hra, ta):
        super().__init__(emp_id, name, designation, department)
        self.basic_salary = basic_salary
        self.da = da
        self.hra = hra
        self.ta = ta

    def calculate_gross_salary(self):
        return self.basic_salary + self.da + self.hra + self.ta

class DeductionSalary(GrossSalary):
    def __init__(self, emp_id, name, designation, department, basic_salary, da, hra, ta, income_tax,
other_deductions):
        super().__init__(emp_id, name, designation, department, basic_salary, da, hra, ta)
        self.income_tax = income_tax
        self.other_deductions = other_deductions

    def calculate_deductions(self):
        return self.income_tax + self.other_deductions

class NetSalary(DeductionSalary):
    def __init__(self, emp_id, name, designation, department, basic_salary, da, hra, ta, income_tax,
other_deductions):
        super().__init__(emp_id, name, designation, department, basic_salary, da, hra, ta, income_tax,
other_deductions)

    def calculate_net_salary(self):
        gross_salary = self.calculate_gross_salary()
        deductions = self.calculate_deductions()
        return gross_salary - deductions

# Example usage:
emp_id = 101
name = "John Doe"
```

```
designation = "Software Engineer"
department = "Engineering"
basic_salary = 50000
da = 0.1 * basic_salary
hra = 0.2 * basic_salary
ta = 0.05 * basic_salary
income_tax = 0.15 * basic_salary
other_deductions = 2000

employee = NetSalary(emp_id, name, designation, department, basic_salary, da, hra, ta,
income_tax, other_deductions)
net_salary = employee.calculate_net_salary()

print(f"Employee ID: {employee.emp_id}")
print(f"Name: {employee.name}")
print(f"Designation: {employee.designation}")
print(f"Department: {employee.department}")
print(f"Net Salary: ${net_salary:.2f}")
```

Output :



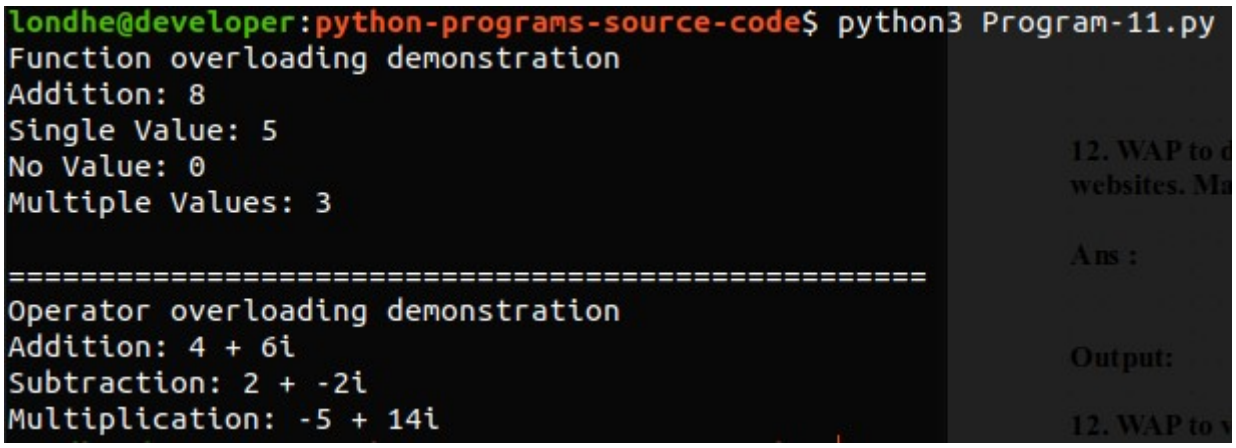
11. WAP to Implement function overloading and operator overloading.

Ans :

```
class MathOperations:
    def add(self, a=None, b=None, *args):
        if a is not None and b is not None:
            return a + b
        elif a is not None and b is None:
            return a
        elif a is None and b is not None:
            return b
        else:
            return sum(args)

# Create an instance of the MathOperations class
math = MathOperations()

# Function overloading demonstration
print("Function overloading demonstration")
print("Addition:", math.add(5, 3))           # Adding two numbers
print("Single Value:", math.add(5))           # Returning single value
print("No Value:", math.add())               # Returning 0
print("Multiple Values:", math.add(1, 2, 3, 4)) # Adding multiple values

class ComplexNumber:
    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary

    def __add__(self, other):
        return ComplexNumber(self.real + other.real, self.imaginary + other.imaginary)

    def __sub__(self, other):
        return ComplexNumber(self.real - other.real, self.imaginary - other.imaginary)

    def __mul__(self, other):
        return ComplexNumber((self.real * other.real) - (self.imaginary * other.imaginary),
                             (self.real * other.imaginary) + (self.imaginary * other.real))

    def __str__(self):
        return f"{self.real} + {self.imaginary}i"

# Create instances of ComplexNumber class
c1 = ComplexNumber(3, 2)
c2 = ComplexNumber(1, 4)

# Operator overloading demonstration
print("\n==================================================")
print("Operator overloading demonstration")
print("Addition:", c1 + c2)  # Addition using overloaded '+'
```

```
print("Subtraction:", c1 - c2)  # Subtraction using overloaded '-'
print("Multiplication:", c1 * c2)  # Multiplication using overloaded '*'
```

Output:

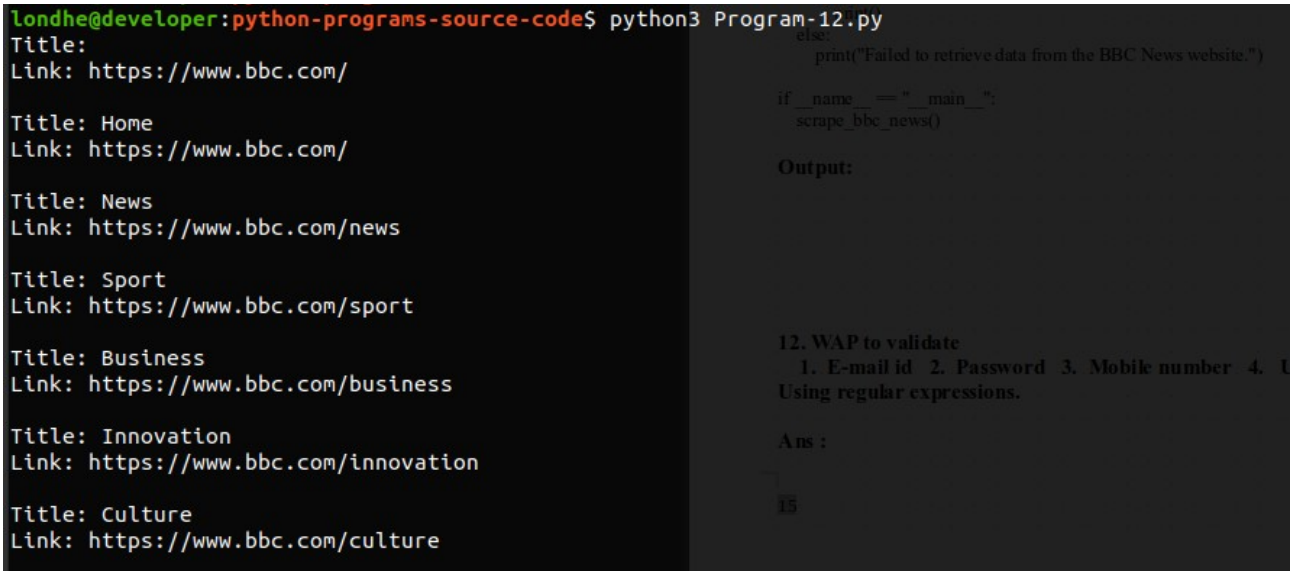


12. WAP to develop simple web scraping application to scrap data from websites. Make a use of required regular expressions.

Ans :

```
import requests
from bs4 import BeautifulSoup
import re

def scrape_bbc_news():
    # URL of the BBC News website
    url = "https://www.bbc.com/news"

    # Send an HTTP GET request to the URL
    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        news_links = soup.find_all('a')

        for link in news_links:
            title = link.text.strip()
            href = link.get('href')
            full_link = f"https://www.bbc.com{href}"

            print(f"Title: {title}")
            print(f"Link: {full_link}")
            print()
    else:
        print("Failed to retrieve data from the BBC News website.")

if __name__ == "__main__":
    scrape_bbc_news()
```

Output:

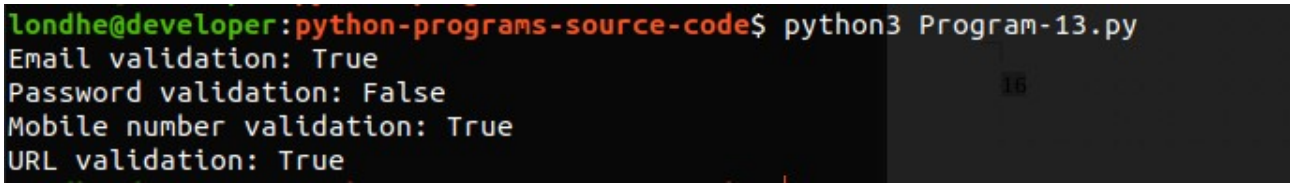


13. WAP to validate
1. E-mail id 2. Password 3. Mobile number 4. URL
Using regular expressions.

Ans :

```
import re

def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    else:
        return False

def validate_password(password):
    pattern = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'
    if re.match(pattern, password):
        return True
    else:
        return False

def validate_mobile_number(mobile_number):
    pattern = r'^\d{10}$'
    if re.match(pattern, mobile_number):
        return True
    else:
        return False

def validate_url(url):
    pattern = r'^(http(s)?://)?(www\.)?[a-zA-Z0-9]+\.[a-zA-Z]{2,}(\.[a-zA-Z]{2,})?$'
    if re.match(pattern, url):
        return True
    else:
        return False


print("Email validation:", validate_email("test@example.com"))
print("Password validation:", validate_password("password@123"))
print("Mobile number validation:", validate_mobile_number("1234567890"))
print("URL validation:", validate_url("https://www.example.com"))
```

Output:



14. WAP to check the age as even or odd by rasing.
Built-in exception and And User defined Exception

Ans :

```
# Custom exception for odd age
class OddAgeError(Exception):
    def __init__(self, age):
        self.age = age

    def __str__(self):
        return f"AgeError: Age '{self.age}' is odd. Please enter an even age."

# Custom exception for even age
class EvenAgeError(Exception):
    def __init__(self, age):
        self.age = age

    def __str__(self):
        return f"AgeError: Age '{self.age}' is even. Please enter an odd age."

def check_age(age):
    if age % 2 == 0:
        raise EvenAgeError(age)
    else:
        raise OddAgeError(age)

try:
    age = int(input("Enter your age: "))
    check_age(age)
except ValueError:
    print("Please enter a valid integer.")
except OddAgeError as odd_error:
    print(odd_error)
except EvenAgeError as even_error:
    print(even_error)
```

Output:

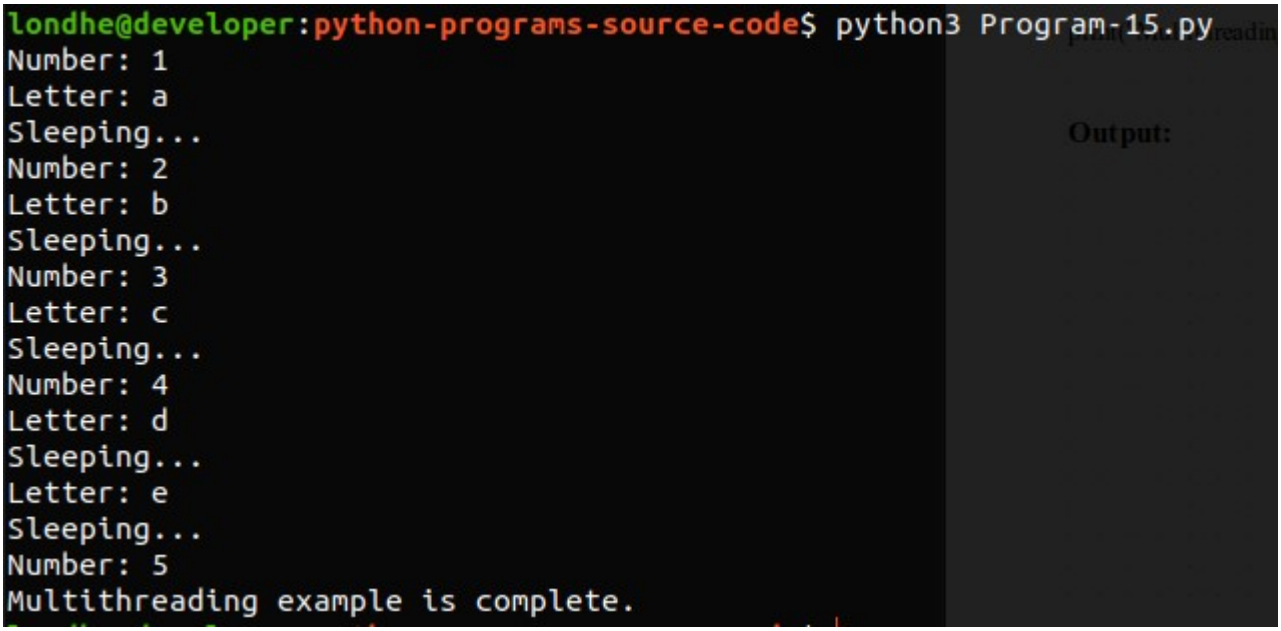
```
londhe@developer:python-programs-source-code$ python3 Program-14.py
Enter your age: 12
AgeError: Age '12' is even. Please enter an odd age.
londhe@developer:python-programs-source-code$ python3 Program-14.py
Enter your age: 55
AgeError: Age '55' is odd. Please enter an even age.
londhe@developer:python-programs-source-code$ python3 Program-14.py
Enter your age: s
Please enter a valid integer.
```

15. Write a Python program to illustrate multithreading.

Ans :

```
import threading
import time

# Function to print numbers from 1 to 5
def print_numbers():
    for i in range(1, 6):
        print(f"Number: {i}")
        time.sleep(1)

# Function to print letters from 'a' to 'e'
def print_letters():
    for char in 'abcde':
        print(f"Letter: {char}")
        time.sleep(1)

# Creating threads for each function
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)

thread1.start()
thread2.start()

thread1.join()
thread2.join()
print("Multithreading example is complete.")
```

Output:



16. Write a program to : 1. Create a new file. 2. Append data 3. Read data Use exception handling to handle IO Error

Ans :

```
def create_file(filename):
    try:
        with open(filename, 'w') as file:
            print(f"File '{filename}' created successfully.")
    except IOError as e:
        print(f"Error creating file '{filename}': {e}")

def append_data(filename, data):
    try:
        with open(filename, 'a') as file:
            file.write(data + '\n')
            print("Data appended to the file.")
    except IOError as e:
        print(f"Error appending data to file '{filename}': {e}")

def read_data(filename):
    try:
        with open(filename, 'r') as file:
            print(f"Data from file '{filename}':")
            print(file.read())
    except IOError as e:
        print(f"Error reading file '{filename}': {e}")

# Main function
def main():
    filename = "example_created-in-program-16.txt"

    # Creating a new file
    create_file(filename)
    data_to_append = "This is some appended data."
    append_data(filename, data_to_append)

    read_data(filename)

if __name__ == "__main__":
    main()
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-16.py
File 'example_created-in-program-16.txt' created successfully.
Data appended to the file.
Data from file 'example_created-in-program-16.txt':
This is some appended data.
```

17. Create Hospital and doctor tables with following fields as- (use MongoDB database)

Hospital Table :-> (Hospital_Id INT UNSIGNED NOT NULL,
Hospital_Name TEXT NOT NULL, Bed_Count INT,
PRIMARY KEY (Hospital_Id)

Doctor table:-> (Doctor_Id INT UNSIGNED NOT NULL,
Doctor_Name TEXT NOT NULL, Hospital_Id INT NOT NULL,
Joining_Date DATE NOT NULL, Speciality TEXT NULL,
Salary INT NULL, Experience INT NULL,
PRIMARY KEY (Doctor_Id)

Executes following

1. **Insert 10 records in both Hospital and doctor tables. Display all the records.**
2. **Fetch Hospital and Doctor Information using hospital Id and doctor Id. (Take a input from user for hospital id and doctor id and display the result accordingly.)**
3. **Fetch all doctors whose salary higher than the input amount and specialty is the same as the input specialty.**
4. **Update doctor experience in years.**

Ans :

```
import pymongo

# Establish a connection to MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create or switch to the database
db = client["hospital_db"]

# Create or switch to the collection for Hospital and Doctor tables
hospital_collection = db["hospital"]
doctor_collection = db["doctor"]

# 1. Insert 10 records in both Hospital and Doctor tables
def insert_records():
    hospitals = [
        {"Hospital_Id": 1, "Hospital_Name": "City Hospital", "Bed_Count": 100},
        {"Hospital_Id": 2, "Hospital_Name": "General Hospital", "Bed_Count": 150},
    ]
    doctors = [
        {"Doctor_Id": 101, "Doctor_Name": "Dr. John", "Hospital_Id": 1, "Joining_Date": "2022-03-
15", "Speciality": "Cardiology", "Salary": 80000, "Experience": 5},
        {"Doctor_Id": 102, "Doctor_Name": "Dr. Emily", "Hospital_Id": 1, "Joining_Date": "2021-07-
20", "Speciality": "Orthopedics", "Salary": 90000, "Experience": 8},
```

```
    ]
    hospital_collection.insert_many(hospitals)
    doctor_collection.insert_many(doctors)

def fetch_hospital_doctor_info(hospital_id, doctor_id):
    hospital_info = hospital_collection.find_one({"Hospital_Id": hospital_id})
    doctor_info = doctor_collection.find_one({"Doctor_Id": doctor_id})
    return hospital_info, doctor_info

def fetch_doctors_by_salary_and_specialty(salary, specialty):
    doctors = doctor_collection.find({"Salary": {"$gt": salary}, "Speciality": specialty})
    return list(doctors)

# 4. Update doctor experience in years
def update_doctor_experience(doctor_id, experience):
    doctor_collection.update_one({"Doctor_Id": doctor_id}, {"$set": {"Experience": experience}})

# Main function
def main():
    insert_records()
    hospital_id = int(input("Enter Hospital ID: "))
    doctor_id = int(input("Enter Doctor ID: "))
    hospital_info, doctor_info = fetch_hospital_doctor_info(hospital_id, doctor_id)
    print("Hospital Information:", hospital_info)
    print("Doctor Information:", doctor_info)

    # 3. Fetch doctors by salary and specialty
    salary = int(input("Enter minimum salary: "))
    specialty = input("Enter specialty: ")
    doctors = fetch_doctors_by_salary_and_specialty(salary, specialty)
    print("Doctors with salary higher than", salary, "and specialty", specialty, ":", doctors)

    # 4. Update doctor experience
    doctor_id = int(input("Enter Doctor ID to update experience: "))
    experience = int(input("Enter new experience in years: "))
    update_doctor_experience(doctor_id, experience)
    print("Doctor experience updated successfully.")

if __name__ == "__main__":
    main()
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-17.py
Enter Hospital ID: 2
Enter Doctor ID: 3
Hospital Information: {'_id': ObjectId('65fc462aecfb13d753eaa46e'), 'Hospital_Id': 2, 'Hospital_Name': 'General Hospital', 'Bed_Count': 150}
Doctor Information: None
Enter minimum salary: 15000
Enter specialty: General
Doctors with salary higher than 15000 and specialty General  : []
Enter Doctor ID to update experience: 3
Enter new experience in years: 12
Doctor experience updated successfully.
```

18. Create a single dimensional array using numpy and executes following commands.

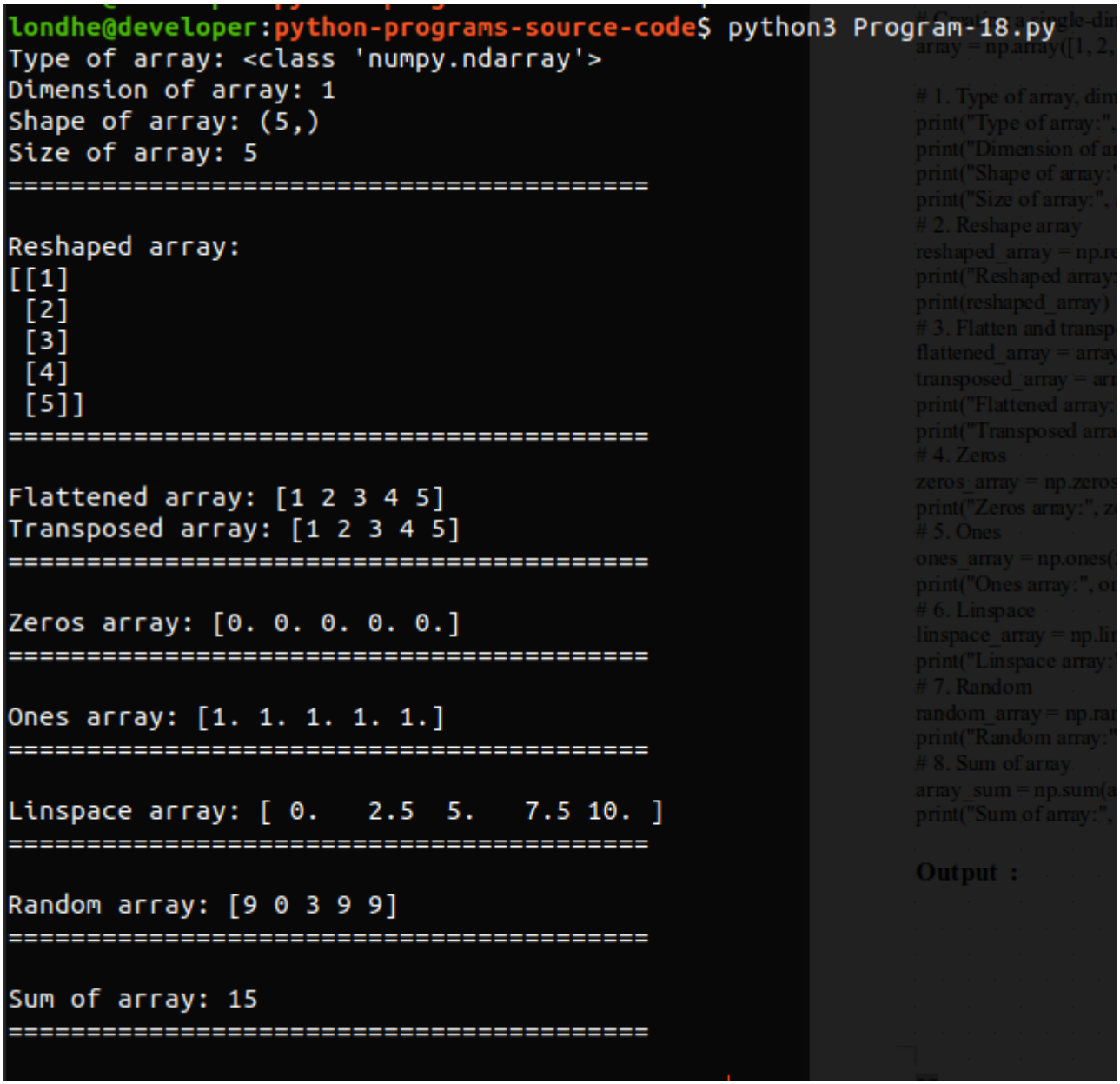
1. **Type of array, dimension of array, shape of array, size of array.**
2. **Reshape array 3. flattern and transpose**
4. **Zeros 5. Ones 6. Linspace 7. random and 8. sum of array.**

Ans :

```
import numpy as np

# Creating a single-dimensional array
array = np.array([1, 2, 3, 4, 5])

# 1. Type of array, dimension of array, shape of array, size of array
print("Type of array:", type(array))
print("Dimension of array:", array.ndim)
print("Shape of array:", array.shape)
print("Size of array:", array.size)
print("========================================\n")

# 2. Reshape array
reshaped_array = np.reshape(array, (5, 1))
print("Reshaped array:")
print(reshaped_array)
print("========================================\n")

# 3. Flatten and transpose
flattened_array = array.flatten()
transposed_array = array.transpose()
print("Flattened array:", flattened_array)
print("Transposed array:", transposed_array)
print("========================================\n")

# 4. Zeros
zeros_array = np.zeros(5)
print("Zeros array:", zeros_array)
print("========================================\n")

# 5. Ones
ones_array = np.ones(5)
print("Ones array:", ones_array)
print("========================================\n")

# 6. Linspace
linspace_array = np.linspace(0, 10, 5)
print("Linspace array:", linspace_array)
print("========================================\n")

# 7. Random
random_array = np.random.randint(0, 10, 5)
```

```
print("Random array:", random_array)
print("==========================================\n")

# 8. Sum of array
array_sum = np.sum(array)
print("Sum of array:", array_sum)
print("==========================================\n")
```

Output :



19. Create array A and array B using arange() and perform following operations.
Addition , Subtraction, Multiplication

Ans :

```
import numpy as np

# Creating arrays A and B using arange()
A = np.arange(1, 6)  # Array A: [1 2 3 4 5]
B = np.arange(6, 11)  # Array B: [6 7 8 9 10]

# Addition
addition_result = A + B
print("Addition:", addition_result)

# Subtraction
subtraction_result = A - B
print("Subtraction:", subtraction_result)

# Multiplication
multiplication_result = A * B
print("Multiplication:", multiplication_result)
```

Output :

20. WAP to Find out space and time utilized by list and array to solve same problem.

Ans :

```
import numpy as np
import time

# Function to find the sum of elements using a list
def sum_using_list(data):
    total = 0
    for num in data:
        total += num
    return total

# Function to find the sum of elements using a NumPy array
def sum_using_array(data):
    return np.sum(data)

# Convert seconds to hours, minutes, seconds, and milliseconds format
def seconds_to_hms(seconds):
    hours = seconds // 3600
    seconds %= 3600
    minutes = seconds // 60
    seconds %= 60
    milliseconds = (seconds - int(seconds)) * 1000
    seconds = int(seconds)
    return hours, minutes, seconds, milliseconds

# Generate a large collection of numbers (for example purposes)
data = list(range(1000000))
array_data = np.array(data)

# Measure time and space utilization for list
start_time = time.time()
list_sum = sum_using_list(data)
end_time = time.time()
list_time_taken = end_time - start_time
list_hours, list_minutes, list_seconds, list_milliseconds = seconds_to_hms(list_time_taken)
list_space_taken = data.__sizeof__() / 1024  # Convert bytes to kilobytes

# Measure time and space utilization for array
start_time = time.time()
array_sum = sum_using_array(array_data)
end_time = time.time()
array_time_taken = end_time - start_time
array_hours, array_minutes, array_seconds, array_milliseconds =
seconds_to_hms(array_time_taken)
array_space_taken = array_data.nbytes / 1024  # Convert bytes to kilobytes

# Display results
```

```
print("Using List:")
print("Time taken:", f"{int(list_hours)}hr, {int(list_minutes):02}min, {int(list_seconds):02}sec,
{int(list_milliseconds):03}ms")
print("Space taken:", f"{list_space_taken:.2f} KB")
print("Sum:", list_sum)
print()

print("Using NumPy Array:")
print("Time taken:", f"{int(array_hours)}hr, {int(array_minutes):02}min,
{int(array_seconds):02}sec, {int(array_milliseconds):03}ms")
print("Space taken:", f"{array_space_taken:.2f} KB")
print("Sum:", array_sum)
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-20.py
Using List:
Time taken: 0hr, 00min, 00sec, 061ms
Space taken: 7812.54 KB
Sum: 499999500000

Using NumPy Array:
Time taken: 0hr, 00min, 00sec, 000ms
Space taken: 7812.50 KB
Sum: 499999500000
```

21. WAP to create a pandas series student and store details like student_id, name, age, mobile and marks.

Ans :

```
import pandas as pd

# Create a list of dictionaries containing student details
students_data = [
    {'student_id': 'S001', 'name': 'John Doe', 'age': 20, 'mobile': '123-456-7890', 'marks': 85},
    {'student_id': 'S002', 'name': 'Jane Smith', 'age': 21, 'mobile': '987-654-3210', 'marks': 90},
    {'student_id': 'S003', 'name': 'Alice Johnson', 'age': 22, 'mobile': '555-555-5555', 'marks': 78},
    {'student_id': 'S004', 'name': 'Bob Brown', 'age': 19, 'mobile': '999-888-7777', 'marks': 95},
    {'student_id': 'S005', 'name': 'Emily Davis', 'age': 20, 'mobile': '333-222-1111', 'marks': 88}
]

# Create a pandas DataFrame from the list of dictionaries
students_df = pd.DataFrame(students_data)

# Display the DataFrame
print("Student Details:")
print(students_df)
```

Output :

```
londhe@developer:python-programs-source-code$ python3 Program-21.py
Student Details:
  student_id           name  age        mobile  marks
0       S001       John Doe   20  123-456-7890     85
1       S002     Jane Smith   21  987-654-3210     90
2       S003  Alice Johnson   22  555-555-5555     78
3       S004      Bob Brown   19  999-888-7777     95
4       S005    Emily Davis   20  333-222-1111     88
```

22. WAP to create a data frame student with fields stude_id, name, class, marks in sub1,sub2,sub3,sub4,sub5, practical, project.

1 Convert data frame into CSV file

2 Load created CSV file

3 Calculate total and add new column using lambda function.

4 Calculate percentage

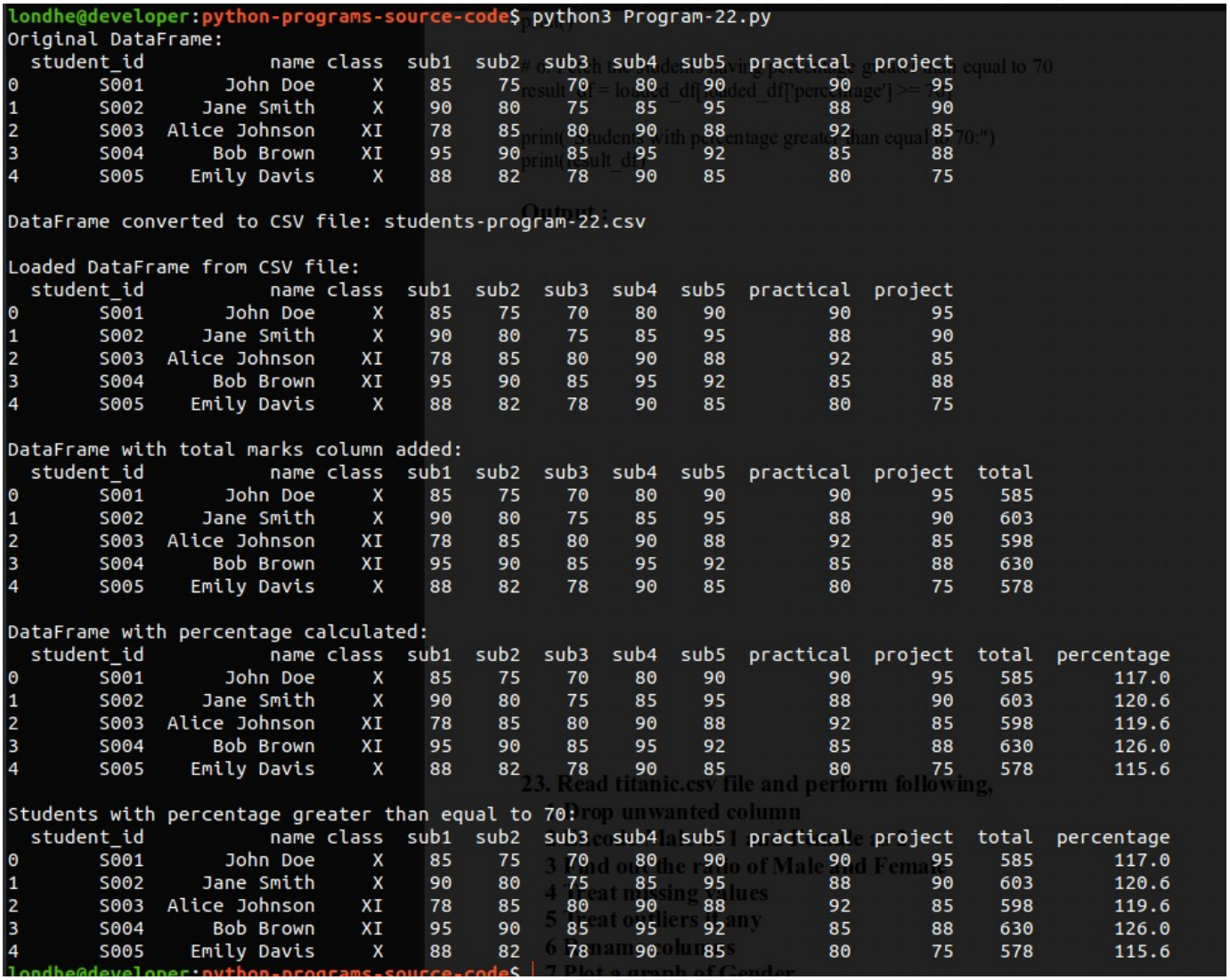
5 Fetch the students having percentage greater than equal to 70

Ans :

```
import pandas as pd

# 1. Create a DataFrame 'student' with fields
student_data = {
    'student_id': ['S001', 'S002', 'S003', 'S004', 'S005'],
    'name': ['John Doe', 'Jane Smith', 'Alice Johnson', 'Bob Brown', 'Emily Davis'],
    'class': ['X', 'X', 'XI', 'XI', 'X'],
    'sub1': [85, 90, 78, 95, 88],
    'sub2': [75, 80, 85, 90, 82],
    'sub3': [70, 75, 80, 85, 78],
    'sub4': [80, 85, 90, 95, 90],
    'sub5': [90, 95, 88, 92, 85],
    'practical': [90, 88, 92, 85, 80],
    'project': [95, 90, 85, 88, 75]
}

student_df = pd.DataFrame(student_data)

# Display the DataFrame
print("Original DataFrame:")
print(student_df)
print()

# 2. Convert DataFrame into CSV file
student_df.to_csv('students-program-22.csv', index=False)
print("DataFrame converted to CSV file: students-program-22.csv")

# 3. Load created CSV file
loaded_df = pd.read_csv('students-program-22.csv')
print("\nLoaded DataFrame from CSV file:")
print(loaded_df)
print()

# 4. Calculate total marks and add new column using lambda function
loaded_df['total'] = loaded_df.apply(lambda row: sum(row[3:]), axis=1)

print("DataFrame with total marks column added:")
print(loaded_df)
print()
```

```
# 5. Calculate percentage
loaded_df['percentage'] = (loaded_df['total'] / 500) * 100

print("DataFrame with percentage calculated:")
print(loaded_df)
print()

# 6. Fetch the students having percentage greater than equal to 70
result_df = loaded_df[loaded_df['percentage'] >= 70]

print("Students with percentage greater than equal to 70:")
print(result_df)
```

Output :



```
londhe@developer:python-programs-source-code$ python3 Program-22.py
Original DataFrame:
  student_id           name class  sub1  sub2  sub3  sub4  sub5  practical  project
0       S001       John Doe     X    85    75    70    80    90         90       95
1       S002     Jane Smith     X    90    80    75    85    95         88       90
2       S003  Alice Johnson    XI    78    85    80    90    88         92       85
3       S004      Bob Brown    XI    95    90    85    95    92         85       88
4       S005    Emily Davis     X    88    82    78    90    85         80       75

DataFrame converted to CSV file: students-program-22.csv

Loaded DataFrame from CSV file:
  student_id           name class  sub1  sub2  sub3  sub4  sub5  practical  project
0       S001       John Doe     X    85    75    70    80    90         90       95
1       S002     Jane Smith     X    90    80    75    85    95         88       90
2       S003  Alice Johnson    XI    78    85    80    90    88         92       85
3       S004      Bob Brown    XI    95    90    85    95    92         85       88
4       S005    Emily Davis     X    88    82    78    90    85         80       75

DataFrame with total marks column added:
  student_id           name class  sub1  sub2  sub3  sub4  sub5  practical  project  total
0       S001       John Doe     X    85    75    70    80    90         90       95    585
1       S002     Jane Smith     X    90    80    75    85    95         88       90    603
2       S003  Alice Johnson    XI    78    85    80    90    88         92       85    598
3       S004      Bob Brown    XI    95    90    85    95    92         85       88    630
4       S005    Emily Davis     X    88    82    78    90    85         80       75    578

DataFrame with percentage calculated:
  student_id           name class  sub1  sub2  sub3  sub4  sub5  practical  project  total  percentage
0       S001       John Doe     X    85    75    70    80    90         90       95    585       117.0
1       S002     Jane Smith     X    90    80    75    85    95         88       90    603       120.6
2       S003  Alice Johnson    XI    78    85    80    90    88         92       85    598       119.6
3       S004      Bob Brown    XI    95    90    85    95    92         85       88    630       126.0
4       S005    Emily Davis     X    88    82    78    90    85         80       75    578       115.6

Students with percentage greater than equal to 70:
  student_id           name class  sub1  sub2  sub3  sub4  sub5  practical  project  total  percentage
0       S001       John Doe     X    85    75    70    80    90         90       95    585       117.0
1       S002     Jane Smith     X    90    80    75    85    95         88       90    603       120.6
2       S003  Alice Johnson    XI    78    85    80    90    88         92       85    598       119.6
3       S004      Bob Brown    XI    95    90    85    95    92         85       88    630       126.0
4       S005    Emily Davis     X    88    82    78    90    85         80       75    578       115.6
londhe@developer:python-programs-source-code$
```

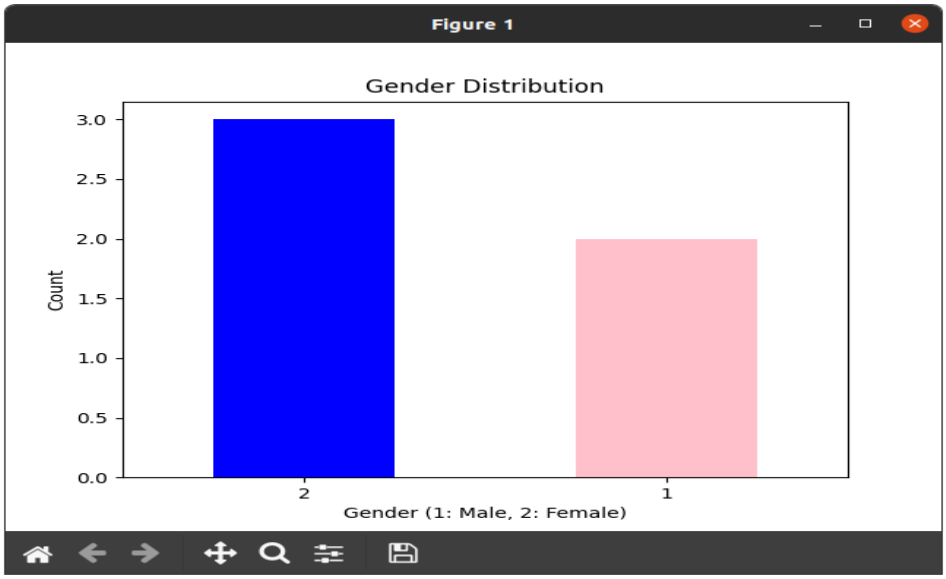
23. Read titanic.csv file and perform following,

1 Drop unwanted column 2 Encode Male as 1 and Female as 2
3 Find out the ratio of Male and Female 4 Treat missing values
5 Treat outliers if any 6 Rename columns 7 Plot a graph of Gender

Ans :

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

titanic_df = pd.read_csv('titanic-program-23.csv')
titanic_df.drop(columns=['Cabin'], inplace=True)
titanic_df['Sex'] = titanic_df['Sex'].map({'male': 1, 'female': 2})

male_count = (titanic_df['Sex'] == 1).sum()
female_count = (titanic_df['Sex'] == 2).sum()
gender_ratio = male_count / female_count
print("Male to Female ratio:", gender_ratio)

titanic_df.fillna(method='ffill', inplace=True)

Q1 = titanic_df['Age'].quantile(0.25)
Q3 = titanic_df['Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
titanic_df = titanic_df[(titanic_df['Age'] >= lower_bound) & (titanic_df['Age'] <= upper_bound)]
titanic_df.rename(columns={'Sex': 'Gender'}, inplace=True)

gender_counts = titanic_df['Gender'].value_counts()
gender_counts.plot(kind='bar', color=['blue', 'pink'])
plt.title('Gender Distribution')
plt.xlabel('Gender (1: Male, 2: Female)')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```

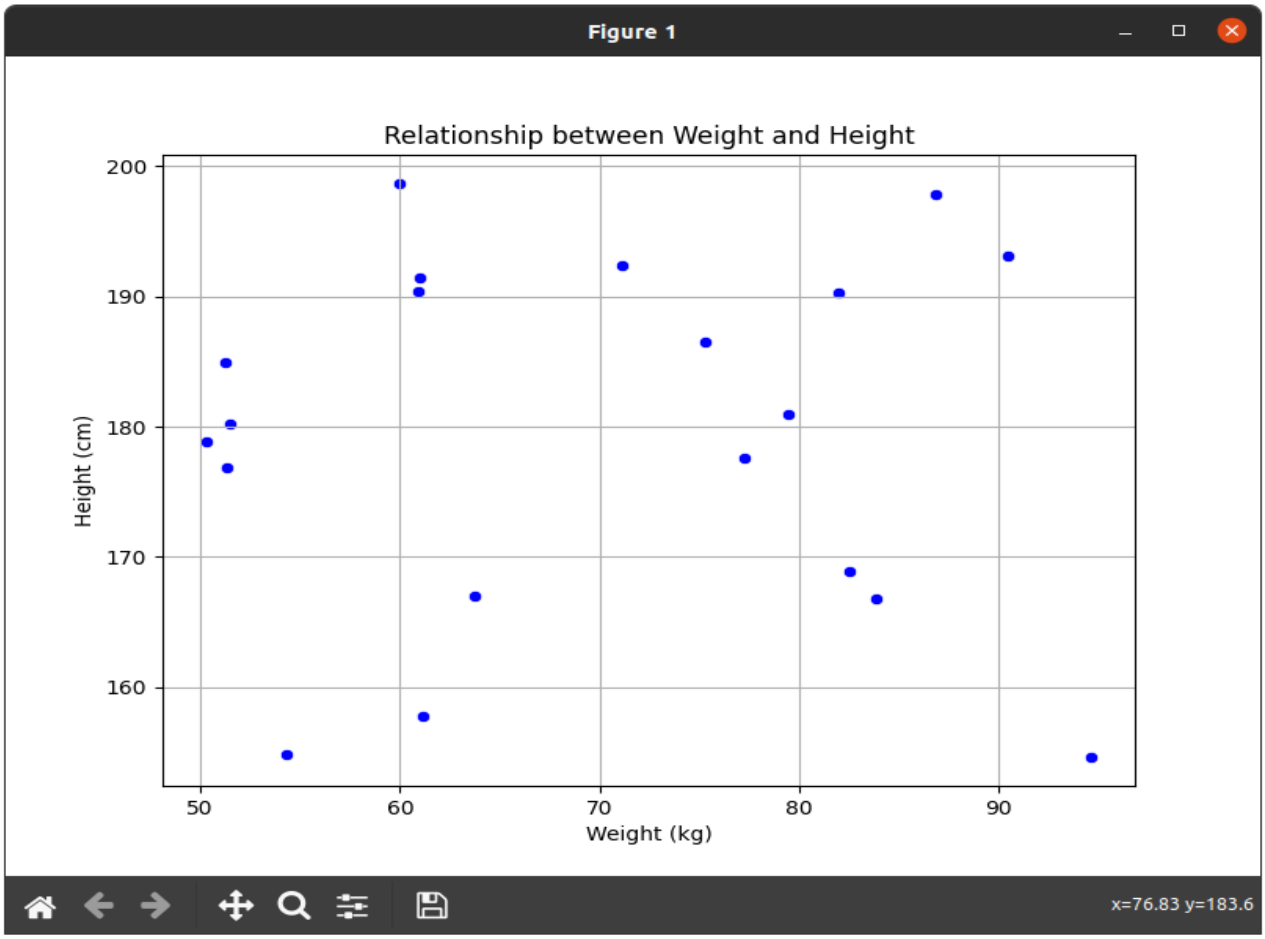
Output :



24. Create lists holding weight and height respectively of 20 person and visualize the relationship using matplotlib and seaborn libraries.
Ans :

```
import matplotlib.pyplot as plt
import seaborn as sns
import random
# Generate random weight and height data for 20 persons
random.seed(42)
weights = [random.uniform(50, 100) for _ in range(20)]
heights = [random.uniform(150, 200) for _ in range(20)]
# Visualize the relationship using matplotlib and seaborn
plt.figure(figsize=(8, 6))
sns.scatterplot(x=weights, y=heights, color='blue')
plt.title('Relationship between Weight and Height')
plt.xlabel('Weight (kg)')
plt.ylabel('Height (cm)')
plt.grid(True)
plt.show()
```

Output :



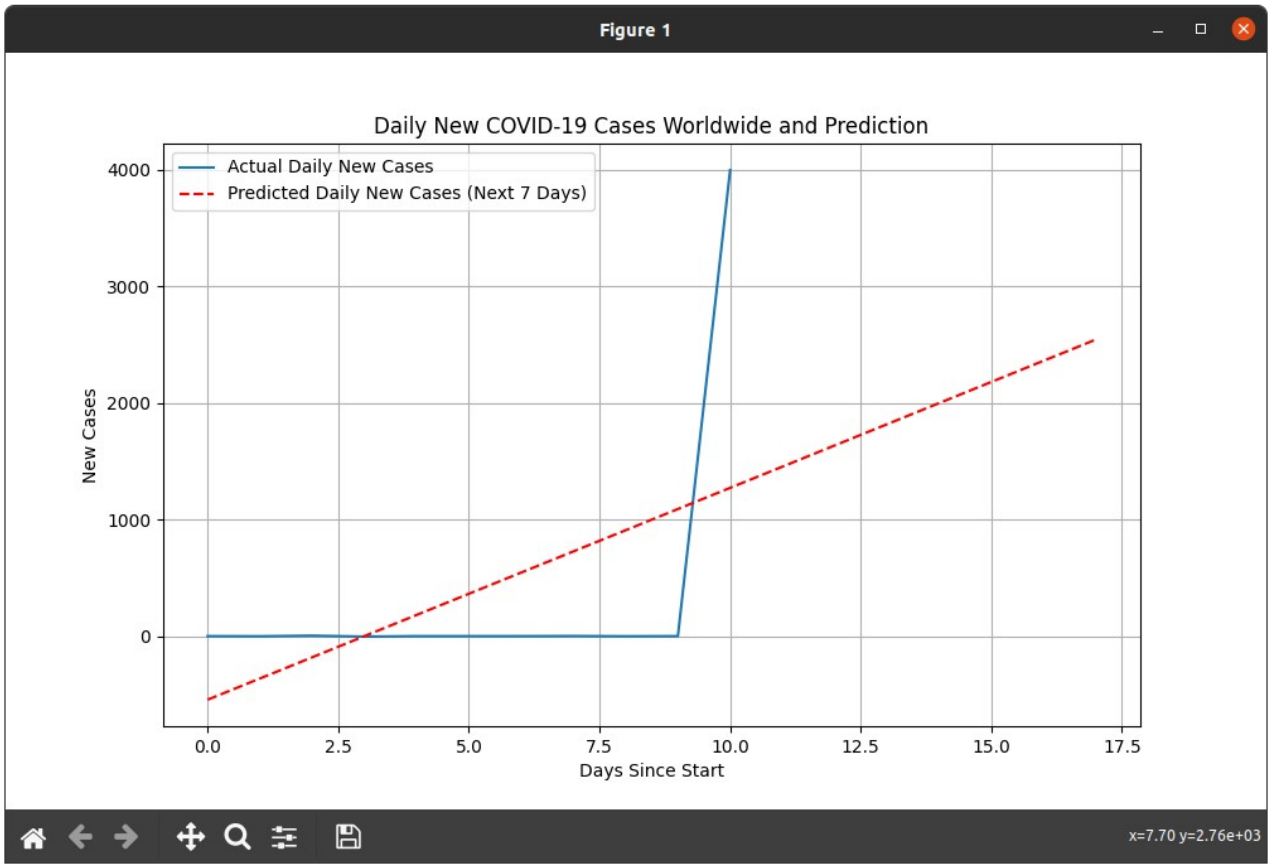
25. Perform Covid-19 data Analysis. (Take current data of world)

Ans :

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import linregress

# Step 1: Data Acquisition
# Get data from following link
# url = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/
csse_covid_19_data/csse_covid_19_daily_reports/02-12-2023.csv"
covid_data = pd.read_csv("covid-sample-data-program-25.csv")

# Step 3: Exploratory Data Analysis (EDA)
covid_data['Date'] = pd.to_datetime(covid_data['Last_Update']).dt.date
daily_new_cases = covid_data.groupby('Date').size().diff().fillna(0)  # Calculate daily new cases
plt.figure(figsize=(10, 6))
plt.plot(daily_new_cases)
plt.title('Daily New COVID-19 Cases Worldwide')
plt.xlabel('Date')
plt.ylabel('New Cases')
plt.grid(True)
plt.show()

# Step 4: Statistical Analysis
total_cases = covid_data['Confirmed'].sum()
total_deaths = covid_data['Deaths'].sum()
total_recovered = covid_data['Recovered'].sum()

print("Total Confirmed Cases Worldwide:", total_cases)
print("Total Deaths Worldwide:", total_deaths)
print("Total Recovered Cases Worldwide:", total_recovered)

# Step 5: Modeling
dates = np.arange(len(daily_new_cases)).reshape(-1, 1)
cases = np.array(daily_new_cases).reshape(-1, 1)
slope, intercept, _, _, _ = linregress(dates.flatten(), cases.flatten())
future_dates = np.arange(len(daily_new_cases) + 7).reshape(-1, 1)
predicted_cases = slope * future_dates + intercept
plt.figure(figsize=(10, 6))
plt.plot(dates, cases, label='Actual Daily New Cases')
plt.plot(future_dates, predicted_cases, linestyle='--', color='red', label='Predicted Daily New Cases
(Next 7 Days)')
plt.title('Daily New COVID-19 Cases Worldwide and Prediction')
plt.xlabel('Days Since Start')
plt.ylabel('New Cases')
plt.legend()
plt.grid(True)
plt.show()
```

```
# Step 6: Interpretation and Conclusion
print("\nInterpretation and Conclusion:")
print("Based on the linear regression model, the predicted number of new cases for the next 7 days is as follows:")
for i in range(1, 8):
    print(f"Day {i}: {predicted_cases[-1 * (7 - i)][0]:.0f}")

print("\nThe linear regression model suggests a trend in the number of new cases, but it's important to note that various factors can influence the actual number of cases, including public health measures, vaccination rates, and virus mutations.")
```

Output :




```
londhe@developer:python-programs-source-code$ python3 Program-25.py
Total Confirmed Cases Worldwide: 672906177
Total Deaths Worldwide: 6856421
Total Recovered Cases Worldwide: 0.0

Interpretation and Conclusion:
Based on the linear regression model, the predicted number of new cases for the next 7 days is as follows:
Day 1: 1636
Day 2: 1818
Day 3: 1999
Day 4: 2181
Day 5: 2363
Day 6: 2545
Day 7: -545

The linear regression model suggests a trend in the number of new cases, but it's important to note that various
al number of cases, including public health measures, vaccination rates, and virus mutations.
```