1	Create a NumPy array from a Python list.

```
import numpy as np

# Python list
python_list = [1, 2, 3, 4, 5]

# Convert Python list to NumPy array
numpy_array = np.array(python_list)

print(numpy_array)
```

2.How do you find the shape of a NumPy array?

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])

# Find the shape of the array
shape = array.shape

print("Shape of the array:", shape)
```

3. How to perform element-wise addition of two NumPy arrays?

```
import numpy as np

# Create two NumPy arrays
array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])

array2 = np.array([[7, 8, 9],
                   [10, 11, 12]])
```

```
# Perform element-wise addition
```

4. Calculate the mean of a NumPy array.

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])

# Calculate the mean of the array
array_mean = np.mean(array)
```

5.Calculate the median of a NumPy array.

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])

# Calculate the median of the array
array_median = np.median(array)

print("Median of the array:", array_median)
```

6.How to find the maximum and minimum values in a NumPy array?

```
import numpy as np

# Create a NumPy array
array = np.array([3, 1, 2, 5, 4])

# Find the maximum value in the array
```

```
max_value = np.amax(array)

# Find the minimum value in the array
min_value = np.amin(array)

print("Maximum value:", max_value)
print("Minimum value:", min_value)
```

7.How to concatenate two NumPy arrays horizontally and vertically?

```
import numpy as np

# Create two NumPy arrays
array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])

array2 = np.array([[7, 8, 9],
                   [10, 11, 12]])

# Concatenate arrays horizontally
horizontal_concatenated = np.concatenate((array1, array2), axis=1)

print("Horizontally concatenated array:")
print(horizontal_concatenated)
```

8. Calculate the dot product of two NumPy arrays.

```
import numpy as np

# Create two NumPy arrays
array1 = np.array([[1, 2],
                   [3, 4]])
```

```
array2 = np.array([[5, 6],
                   [7, 8]])

# Calculate the dot product
dot_product = np.dot(array1, array2)

print("Dot product of the arrays:")
print(dot_product)
```

9. Find the unique elements and their counts in a NumPy array.

```
import numpy as np

# Create a NumPy array
array = np.array([1, 2, 3, 1, 2, 4, 5, 1])

# Find the unique elements and their counts
unique_elements, counts = np.unique(array,
return_counts=True)

# Zip the unique elements and their counts together
unique_with_counts = list(zip(unique_elements, counts))

print("Unique elements and their counts:")
print(unique_with_counts)
```

10. Create a NumPy array with elements 1 to 10.

```
import numpy as np

# Create a NumPy array with elements 1 to 10
array = np.arange(1, 11)

print("NumPy array with elements 1 to 10:")
```

```
print(array)
```

11. Create a 3x3 identity matrix using NumPy

```
import numpy as np

# Create a 3x3 identity matrix
identity_matrix = np.identity(3)

print("3x3 Identity Matrix:")
print(identity_matrix)
```

12. Create a NumPy array with a specified upper and lower limit.

```
import numpy as np

# Specify the upper and lower limit
lower_limit = 1
upper_limit = 10

# Create a NumPy array with values from lower_limit to upper_limit
array = np.linspace(lower_limit, upper_limit, num=10)

print("NumPy array with specified upper and lower limit:")
print(array)
```

13. Calculate the sum of all elements in a NumPy array.

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])
```

```
# Calculate the sum of all elements in the array
array_sum = np.sum(array)

print("Sum of all elements in the array:", array_sum)
```

14. Replace all even numbers in a NumPy array with 0.

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])

# Replace even numbers with 0
array[array % 2 == 0] = 0

print("Array with even numbers replaced by 0:")
print(array)
```

15. Convert a NumPy array to a Python list.

```
import numpy as np


# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])


# Convert the NumPy array to a Python list
list_from_array = array.tolist()
```

```
print("Python list converted from NumPy array:")
print(list_from_array)
```

16. Calculate the inverse of a square NumPy matrix.

```
import numpy as np

# Create a square NumPy matrix
matrix = np.array([[1, 2],
                   [3, 4]])

# Calculate the inverse of the matrix
inverse_matrix = np.linalg.inv(matrix)

print("Inverse of the matrix:")
print(inverse_matrix)
```

17. Remove all NaN values from a NumPy array.

```
import numpy as np

# Create a NumPy array with NaN values
array = np.array([1, 2, np.nan, 4, np.nan, 6])

# Remove NaN values from the array
array_without_nan = array[~np.isnan(array)]
```

```
print("Array without NaN values:")
print(array_without_nan)
```

18.Perform element-wise subtraction of two NumPy arrays.

```
import numpy as np

# Create two NumPy arrays
array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])

array2 = np.array([[7, 8, 9],
                   [10, 11, 12]])

# Perform element-wise subtraction
result_array = array1 - array2

print("Result of element-wise subtraction:")
print(result_array)
```

19 Perform element-wise division of two NumPy arrays.

```
import numpy as np

# Create two NumPy arrays
```

```
array1 = np.array([[10, 20, 30],
                   [40, 50, 60]])

array2 = np.array([[2, 4, 6],
                   [8, 10, 12]])

# Perform element-wise division
result_array = array1 / array2

print("Result of element-wise division:")
print(result_array)
```

20. Find the indices of the minimum and maximum values in a NumPy array.

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])

# Find the index of the minimum value
min_index = np.argmin(array)
# Find the index of the maximum value
max_index = np.argmax(array)
```

```
# Convert the 1D index to 2D indices
min_indices = np.unravel_index(min_index, array.shape)
max_indices = np.unravel_index(max_index, array.shape)

print("Index of the minimum value:", min_indices)
print("Index of the maximum value:", max_indices)
```

21. Check if two NumPy arrays are equal.

```
import numpy as np

# Create two NumPy arrays
array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])

array2 = np.array([[1, 2, 3],
                   [4, 5, 6]])

# Check if the arrays are equal
are_equal = np.array_equal(array1, array2)

print("Are the arrays equal?", are_equal)
```

22 Extract specific rows and columns from a NumPy array.

```
import numpy as np

# Create a NumPy array
```

```
array = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

# Extract specific rows (e.g., rows 0 and 2) and columns (e.g., columns 1 and 2)
selected_rows = array[[0, 2], :]  # Rows 0 and 2, all columns
selected_columns = array[:, [1, 2]]  # All rows, columns 1 and 2

print("Selected rows:")
print(selected_rows)

print("\nSelected columns:")
print(selected_columns)
```

23. Sort a NumPy array in ascending order.

```
import numpy as np

# Create a NumPy array
array = np.array([3, 1, 2, 5, 4])

# Sort the array in ascending order
sorted_array = np.sort(array)
```

```
print("Sorted array in ascending order:")
print(sorted_array)
```

24, Sort a NumPy array in descending order.

```
import numpy as np

# Create a NumPy array
array = np.array([3, 1, 2, 5, 4])

# Sort the array in descending order
sorted_array_desc = np.sort(array)[::-1]

print("Sorted array in descending order:")
print(sorted_array_desc)
```

25, Round the elements of a NumPy array to the nearest integer.

```
import numpy as np

# Create a NumPy array
array = np.array([1.1, 2.5, 3.9, 4.6])

# Round the elements to the nearest integer
rounded_array = np.round(array)
```

```
print("Rounded array to the nearest integer:")
print(rounded_array)
```

26. Check if any element in a NumPy array is NaN.

```
import numpy as np

# Create a NumPy array
array = np.array([1, 2, np.nan, 4, 5])

# Check if any element in the array is NaN
has_nan = np.any(np.isnan(array))

if has_nan:
    print("The array contains NaN.")
else:
    print("The array does not contain NaN.")
```

27. Create a NumPy array and print its size, and data type.

```
import numpy as np

# Create a NumPy array
array = np.array([[1, 2, 3],
                  [4, 5, 6]])

# Print the size of the array
print("Size of the array:", array.size)
```

```
# Print the data type of the array
print("Data type of the array:", array.dtype)
```

28 Write a Python program to print a pyramid pattern

```
def print_pyramid(rows):
    for i in range(1, rows + 1):
        # Print spaces
        print(" " * (rows - i), end="")
        # Print stars
        print("* " * i)

# Number of rows in the pyramid
rows = 5
print("Pyramid pattern:")
print_pyramid(rows)
```

29. Create a Python program to print a diamond pattern with a given number of rows.

```
def print_diamond(rows):
    # Upper half of the diamond
    for i in range(1, rows + 1):
        print(" " * (rows - i) + "* " * i)
    # Lower half of the diamond (excluding the middle row if rows is odd)
    for i in range(rows - 1, 0, -1):
```

```
        print(" " * (rows - i) + "* " * i)


# Number of rows in the diamond
rows = 5
if rows % 2 == 0:
    print("Diamond pattern with", rows, "rows (excluding the middle row):")
else:
    print("Diamond pattern with", rows, "rows:")
print_diamond(rows)
```

30. Write a Python program to print a pyramid pattern with numbers, where each row contains a sequence of numbers starting from 1 and incrementing by 1.

```
def print_number_pyramid(rows):
    # Loop through each row
    for i in range(1, rows + 1):
        # Print spaces
        print(" " * (rows - i), end="")
        # Print numbers in ascending order
        for j in range(1, i + 1):
            print(j, end=" ")
        # Move to the next line
        print()
```

```
# Number of rows in the pyramid
rows = 5
print("Number pyramid pattern:")
print_number_pyramid(rows)
```

31. Create a Python program to check if a given number is an Adam number, and provide the necessary output based on the condition. (range: 100 - 1000)

```
def is_adam_number(number):
    # Square the number
    square = number ** 2
    # Reverse the digits of the square
    reverse_square = int(str(square)[::-1])
    # Square the reverse square
    reverse_square_square = reverse_square ** 2
    # Check if the original square and reverse square squared are the same
    return square == int(str(reverse_square_square)[::-1])

# Check for Adam numbers in the range 100 to 1000
adam_numbers = []
for num in range(100, 1001):
    if is_adam_number(num):
        adam_numbers.append(num)
```

```
# Print the Adam numbers found
if len(adam_numbers) > 0:
    print("Adam numbers within the range 100 to 1000:")
    print(adam_numbers)
else:
    print("No Adam numbers found within the range 100 to 1000.")
```

32. Write a Python program that checks if a given number is an automorphic number and provides the output accordingly. (range: 1 - 1000)

```
def is_automorphic(number):
    # Square the number
    square = number ** 2
    # Convert the number and its square to strings
    num_str = str(number)
    square_str = str(square)
    # Check if the last digits of the square match the number itself
    return square_str.endswith(num_str)


# Check for automorphic numbers in the range 1 to 1000
automorphic_numbers = []
for num in range(1, 1001):
    if is_automorphic(num):
```

```
        automorphic_numbers.append(num)


# Print the automorphic numbers found
if len(automorphic_numbers) > 0:
    print("Automorphic numbers within the range 1 to 1000:")
    print(automorphic_numbers)
else:
    print("No automorphic numbers found within the range 1 to 1000.")
```

33 Develop a Python program to find and display all perfect numbers within a given range. (range: 1 - 100)

```
def is_perfect_number(number):
    # Find the sum of proper divisors
    divisor_sum = sum([divisor for divisor in range(1, number) if number % divisor == 0])
    # Check if the sum of proper divisors equals the number
    return divisor_sum == number


# Check for perfect numbers in the range 1 to 100
perfect_numbers = []
for num in range(1, 101):
    if is_perfect_number(num):
        perfect_numbers.append(num)
```

```
# Print the perfect numbers found
if len(perfect_numbers) > 0:
    print("Perfect numbers within the range 1 to 100:")
    print(perfect_numbers)
else:
    print("No perfect numbers found within the range 1 to 100.")
```

34, Create a Python program to determine if a given number is a happy number or not. Provide output based on the result. (range: 1 - 100)

```
def is_happy_number(number):
    def get_next_number(n):
        next_number = 0
        while n > 0:
            digit = n % 10
            next_number += digit ** 2
            n //= 10
        return next_number

    seen = set()
    while number != 1 and number not in seen:
        seen.add(number)
        number = get_next_number(number)
    return number == 1
```

```
# Check for happy numbers in the range 1 to 100
happy_numbers = []
for num in range(1, 101):
    if is_happy_number(num):
        happy_numbers.append(num)

# Print the happy numbers found
if len(happy_numbers) > 0:
    print("Happy numbers within the range 1 to 100:")
    print(happy_numbers)
else:
    print("No happy numbers found within the range 1 to 100.")
```

35, Write a Python program that checks if a given number is an Armstrong number, and provide the necessary output based on the condition. (range: 100 - 1000) [An armstrong number is any number that is equal to the sum of cube of its individual digits]

```
def is_armstrong_number(number):
    # Calculate the number of digits
    num_digits = len(str(number))
    # Calculate the sum of cubes of digits
    sum_cubes = sum(int(digit) ** num_digits for digit in str(number))
```

```
    # Check if the number is equal to the sum of cubes
    return sum_cubes == number

# Check for Armstrong numbers in the range 100 to 1000
armstrong_numbers = []
for num in range(100, 1001):
    if is_armstrong_number(num):
        armstrong_numbers.append(num)

# Print the Armstrong numbers found
if len(armstrong_numbers) > 0:
    print("Armstrong numbers within the range 100 to 1000:")
    print(armstrong_numbers)
else:
    print("No Armstrong numbers found within the range 100
to 1000.")
```

36 Develop an employee management system. Create an abstract "Employee" class with abstract methods for calculating salary and displaying information. Implement subclasses for various roles, like "Manager," "Developer," and "Designer," with role-specific implementations.

```
from abc import ABC, abstractmethod

class Employee(ABC):
    def __init__(self, name, employee_id):
```

```
        self.name = name
        self.employee_id = employee_id

    @abstractmethod
    def calculate_salary(self):
        pass

    @abstractmethod
    def display_information(self):
        pass

class Manager(Employee):
    def __init__(self, name, employee_id, salary, bonus):
        super().__init__(name, employee_id)
        self.salary = salary
        self.bonus = bonus

    def calculate_salary(self):
        return self.salary + self.bonus

    def display_information(self):
        print(f"Name: {self.name}")
        print(f"Employee ID: {self.employee_id}")
```

```
        print(f"Role: Manager")
        print(f"Salary: ${self.calculate_salary()}")

class Developer(Employee):
    def __init__(self, name, employee_id, salary, language):
        super().__init__(name, employee_id)
        self.salary = salary
        self.language = language

    def calculate_salary(self):
        return self.salary

    def display_information(self):
        print(f"Name: {self.name}")
        print(f"Employee ID: {self.employee_id}")
        print(f"Role: Developer")
        print(f"Salary: ${self.calculate_salary()}")
        print(f"Programming Language: {self.language}")

class Designer(Employee):
    def __init__(self, name, employee_id, salary, tools):
        super().__init__(name, employee_id)
        self.salary = salary
```

```
        self.tools = tools

    def calculate_salary(self):
        return self.salary

    def display_information(self):
        print(f"Name: {self.name}")
        print(f"Employee ID: {self.employee_id}")
        print(f"Role: Designer")
        print(f"Salary: ${self.calculate_salary()}")
        print(f"Design Tools: {self.tools}")

# Example usage:
manager = Manager("John Doe", 1001, 60000, 10000)
developer = Developer("Jane Smith", 1002, 50000, "Python")
designer = Designer("Alice Johnson", 1003, 55000,
["Photoshop", "Illustrator"])

manager.display_information()
print("\n")
developer.display_information()
print("\n")
designer.display_information()
```

37 Create a banking operations system. Develop an abstract "BankAccount" class with abstract methods for deposit and withdrawal. Implement concrete subclasses for "SavingsAccount" and "CheckingAccount" with their transaction handling.

```
from abc import ABC, abstractmethod


from abc import ABC, abstractmethod


class BankAccount(ABC):
    def __init__(self, account_number, balance=0):
        self.account_number = account_number
        self.balance = balance


    @abstractmethod
    def deposit(self, amount):
        pass


    @abstractmethod
    def withdraw(self, amount):
        pass


class SavingsAccount(BankAccount):
    def __init__(self, account_number, balance=0,
interest_rate=0.02):
```

```
        super().__init__(account_number, balance)
        self.interest_rate = interest_rate

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

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrew ${amount}. Current balance:
${self.balance}")
        else:
            print("Insufficient funds.")

    def calculate_interest(self):
        interest_amount = self.balance * self.interest_rate
        self.balance += interest_amount
        print(f"Interest added: ${interest_amount}. Current
balance: ${self.balance}")

class CheckingAccount(BankAccount):
```

```
    def __init__(self, account_number, balance=0, overdraft_limit=100):
        super().__init__(account_number, balance)
        self.overdraft_limit = overdraft_limit

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

    def withdraw(self, amount):
        if self.balance + self.overdraft_limit >= amount:
            self.balance -= amount
            print(f"Withdrew ${amount}. Current balance: ${self.balance}")
        else:
            print("Transaction declined. Overdraft limit exceeded.")

# Example usage:
savings_account = SavingsAccount("SA123", balance=1000)
checking_account = CheckingAccount("CA456", balance=500, overdraft_limit=200)
```

```
savings_account.deposit(500)
savings_account.calculate_interest()
savings_account.withdraw(200)

print("\n")

checking_account.deposit(300)
checking_account.withdraw(700)
checking_account.withdraw(300)
```

38 Create a text processing tool that formats text. Define an abstract class "TextProcessor" with abstract methods for formatting and analyzing text. Implement concrete subclasses like "UpperCaseFormatter" and "LowerCaseFormatter" to modify text as needed.

```
from abc import ABC, abstractmethod

class TextProcessor(ABC):
    def __init__(self, text):
        self.text = text

    @abstractmethod
    def format_text(self):
        pass

    @abstractmethod
```

```
    def analyze_text(self):
        pass

class UpperCaseFormatter(TextProcessor):
    def format_text(self):
        return self.text.upper()

    def analyze_text(self):
        word_count = len(self.text.split())
        return f"Number of words: {word_count}"

class LowerCaseFormatter(TextProcessor):
    def format_text(self):
        return self.text.lower()

    def analyze_text(self):
        character_count = len(self.text)
        return f"Number of characters: {character_count}"

# Example usage:
text = "Hello World! This is a Text Processing Tool."

upper_case_formatter = UpperCaseFormatter(text)
```

```
lower_case_formatter = LowerCaseFormatter(text)

print("Original text:")
print(text)

print("\nFormatted text (upper case):")
formatted_text_upper = upper_case_formatter.format_text()
print(formatted_text_upper)

print("\nAnalysis:")
analysis_upper = upper_case_formatter.analyze_text()
print(analysis_upper)

print("\nFormatted text (lower case):")
formatted_text_lower = lower_case_formatter.format_text()
print(formatted_text_lower)

print("\nAnalysis:")
analysis_lower = lower_case_formatter.analyze_text()
print(analysis_lower)
```

39 Design a program for calculating the areas of geometric shapes. Create an abstract "Shape" class with abstract methods for calculating area and perimeter. Define subclasses for

"Circle" and "Rectangle" and provide their area calculation methods.

```
from abc import ABC, abstractmethod
import math

class Shape(ABC):
    @abstractmethod
    def calculate_area(self):
        pass

    @abstractmethod
    def calculate_perimeter(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius**2

    def calculate_perimeter(self):
        return 2 * math.pi * self.radius
```

```
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.width * self.height

    def calculate_perimeter(self):
        return 2 * (self.width + self.height)

# Example usage:
circle = Circle(5)
rectangle = Rectangle(4, 6)

print("Circle:")
print("Area:", circle.calculate_area())
print("Perimeter:", circle.calculate_perimeter())

print("\nRectangle:")
print("Area:", rectangle.calculate_area())
print("Perimeter:", rectangle.calculate_perimeter())
```

40 Design a class hierarchy for an online shopping system. Create a base class "Product" and subclasses like "Electronics," "Clothing," and "Books." Each subclass should have methods to calculate shipping costs and provide product details.

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def calculate_shipping_cost(self, quantity):
        # Default shipping cost calculation logic
        return 0

    def product_details(self):
        return f"Name: {self.name}\nPrice: ${self.price:.2f}"

class Electronics(Product):
    def __init__(self, name, price, weight):
        super().__init__(name, price)
        self.weight = weight

    def calculate_shipping_cost(self, quantity):
        # Example shipping cost calculation based on weight and quantity
```

```
        return 3 * self.weight * quantity

    def product_details(self):
        details = super().product_details()
        return f"{details}\nWeight: {self.weight} kg"

class Clothing(Product):
    def __init__(self, name, price, size):
        super().__init__(name, price)
        self.size = size

    def calculate_shipping_cost(self, quantity):
        # Example shipping cost calculation based on size and
quantity
        return 2 * len(self.size) * quantity

    def product_details(self):
        details = super().product_details()
        return f"{details}\nSize: {self.size}"

class Books(Product):
    def __init__(self, name, price, author):
        super().__init__(name, price)
```

```
        self.author = author

    def calculate_shipping_cost(self, quantity):
        # Example shipping cost calculation based on weight and quantity
        return 1.5 * quantity

    def product_details(self):
        details = super().product_details()
        return f"{details}\nAuthor: {self.author}"

# Example usage:
electronics_product = Electronics("Laptop", 999.99, 2.5)
clothing_product = Clothing("T-shirt", 19.99, "M")
book_product = Books("Python Programming", 29.99, "Guido van Rossum")

print("Electronics Product:")
print(electronics_product.product_details())
print("Shipping Cost for 2 items:", electronics_product.calculate_shipping_cost(2))

print("\nClothing Product:")
print(clothing_product.product_details())
```

```
print("Shipping Cost for 3 items:",
clothing_product.calculate_shipping_cost(3))


print("\nBook Product:")
print(book_product.product_details())
print("Shipping Cost for 1 item:",
book_product.calculate_shipping_cost(1))
```

41. Write a Python program that calculates the factorial of a given number using recursion.

```
def factorial(n):
    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    # Recursive case: factorial of n is n multiplied by factorial of n-1
    else:
        return n * factorial(n - 1)


# Example usage:
number = 5
print(f"The factorial of {number} is:", factorial(number))
```

42 Write a Python program that calculates the factorial of a given number without using recursion.

```
def factorial(n):
    result = 1
```

```
    for i in range(1, n + 1):
        result *= i
    return result


# Example usage:
number = 5
print(f"The factorial of {number} is:", factorial(number))
```

43 Write a Python function that takes an integer as input and determines whether it is a prime number or not. { A prime number is a number greater than 1 that has that has no divisors other than 1 and itself. }

```
def is_prime(n):
    # Prime numbers are greater than 1
    if n <= 1:
        return False
    # Check for factors from 2 to square root of n
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True


# Example usage:
number = 17
if is_prime(number):
```

```
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

44 Create a number guessing game where the computer generates a random number, and the player has to guess it. Use a combination of while and for loops to implement the game with attempts and hints.

```
import random


def number_guessing_game():
    # Generate a random number between 1 and 100
    secret_number = random.randint(1, 100)
    attempts = 0


    print("Welcome to the Number Guessing Game!")
    print("I've picked a number between 1 and 100. Can you guess it?")


    while True:
        guess = int(input("\nEnter your guess (1-100): "))
        attempts += 1


        if guess < secret_number:
            print("Too low! Try guessing higher.")
        elif guess > secret_number:
```

```
            print("Too high! Try guessing lower.")
        else:
            print(f"Congratulations! You guessed the number {secret_number} correctly!")
            print(f"It took you {attempts} attempts.")
            break

        if attempts % 3 == 0:
            print("Hint: The number is between", secret_number - 10, "and", secret_number + 10)

# Play the game
number_guessing_game()
```

45 Write a Python program to find all prime numbers within a given range using a for loop

```
def find_primes_in_range(start, end):
    prime_numbers = []
    for num in range(start, end + 1):
        if num > 1:
            is_prime = True
            for i in range(2, int(num**0.5) + 1):
                if num % i == 0:
                    is_prime = False
                    break
```

```
            if is_prime:
                prime_numbers.append(num)
    return prime_numbers


# Example usage:
start_range = 10
end_range = 50
print(f"Prime numbers between {start_range} and {end_range}:")
print(find_primes_in_range(start_range, end_range))
```

46. Implement a program to encrypt a string by shifting each character by a certain number of positions in the alphabet.

```
def encrypt_string(text, shift):
    encrypted_text = ""
    for char in text:
        # Encrypt uppercase characters
        if char.isupper():
            encrypted_text += chr((ord(char) - 65 + shift) % 26 + 65)
        # Encrypt lowercase characters
        elif char.islower():
            encrypted_text += chr((ord(char) - 97 + shift) % 26 + 97)
        # Leave non-alphabetic characters unchanged
```

```
        else:
            encrypted_text += char
    return encrypted_text

# Example usage:
text = "Hello World!"
shift = 3
encrypted_text = encrypt_string(text, shift)
print(f"Original text: {text}")
print(f"Encrypted text (shifted by {shift} positions): {encrypted_text}")
```

47. Write a program that performs operations on a list, such as adding elements, removing duplicates, and sorting.

```
def list_operations(input_list):
    # Add elements to the list
    input_list.extend([8, 2, 5])
    print("List after adding elements:", input_list)

    # Remove duplicates from the list
    input_list = list(set(input_list))
    print("List after removing duplicates:", input_list)

    # Sort the list in ascending order
    input_list.sort()
```

```
    print("List after sorting in ascending order:", input_list)

    # Sort the list in descending order
    input_list.sort(reverse=True)
    print("List after sorting in descending order:", input_list)

# Example usage:
my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
print("Original list:", my_list)
list_operations(my_list.copy())
```

48 Write a Python program that takes a sentence as input from the user and counts the number of words in the sentence. Words are separated by spaces.

```
def count_words(sentence):
    # Split the sentence into words using whitespace as delimiter
    words = sentence.split()
    # Count the number of words
    num_words = len(words)
    return num_words

# Take input from the user
sentence = input("Enter a sentence: ")
```

```
# Call the count_words function and display the result
word_count = count_words(sentence)
print("Number of words in the sentence:", word_count)
```

49 Develop a program that filters a list of numbers to create two separate lists, one containing even numbers and the other containing odd numbers.

```
def filter_even_odd(numbers):
    even_numbers = []
    odd_numbers = []
    for num in numbers:
        if num % 2 == 0:
            even_numbers.append(num)
        else:
            odd_numbers.append(num)
    return even_numbers, odd_numbers


# Example usage:
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers, odd_numbers = filter_even_odd(numbers)
print("Original list:", numbers)
print("Even numbers:", even_numbers)
print("Odd numbers:", odd_numbers)
```

50 Write a program that performs operations on tuples, including concatenation, indexing, and slicing.

```
def tuple_operations(tuple1, tuple2):
```

```
    # Concatenation of tuples
    concatenated_tuple = tuple1 + tuple2
    print("Concatenated tuple:", concatenated_tuple)

    # Indexing
    print("First element of tuple1:", tuple1[0])
    print("Last element of tuple2:", tuple2[-1])

    # Slicing
    print("Sliced portion of concatenated tuple:",
concatenated_tuple[2:5])


# Example usage:
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
print("Tuple 1:", tuple1)
print("Tuple 2:", tuple2)
tuple_operations(tuple1, tuple2)
```

51 Write a program that must read a CSV file, display the second column, and increment the values in the third column by a fixed amount (e.g., 10).

```
import csv

def process_csv_file(file_name, increment):
    try:
```

```
        # Open the CSV file
        with open(file_name, 'r') as file:
            # Create a CSV reader object
            csv_reader = csv.reader(file)
            # Skip the header row
            next(csv_reader)
            # Iterate over each row in the CSV file
            for row in csv_reader:
                # Display the second column
                print("Second column value:", row[1])
                # Increment the value in the third column by the
specified amount
                try:
                    updated_value = int(row[2]) + increment
                    print("Updated value in the third column:",
updated_value)
                except ValueError:
                    print("Error: Value in the third column is not an
integer.")
    except FileNotFoundError:
        print("Error: File not found.")

# Example usage:
file_name = 'example.csv'  # Change this to the name of your
CSV file or provide the correct path
increment_amount = 10
process_csv_file(file_name, increment_amount)
```

52 Create a program that reads a CSV file, filters rows based on a specific condition in one column (e.g., values greater than 50), and then multiplies the values in another column by a factor (e.g., 1.5).

```
import csv
```

```
def process_csv_file(file_name, condition_column_index, condition_value, factor_column_index, factor):
    filtered_rows = []
    # Open the CSV file
    with open(file_name, 'r') as file:
        # Create a CSV reader object
        csv_reader = csv.reader(file)
        # Skip the header row
        header = next(csv_reader)
        filtered_rows.append(header)
        # Iterate over each row in the CSV file
        for row in csv_reader:
            # Check if the condition in the specified column is met
            if int(row[condition_column_index]) > condition_value:
                # Multiply the value in the specified column by the factor
                row[factor_column_index] = str(float(row[factor_column_index]) * factor)
                filtered_rows.append(row)
    return filtered_rows


# Example usage:
```

```
file_name = 'example.csv'  # Change this to the name of your CSV file
condition_column_index = 2
condition_value = 50
factor_column_index = 3
factor = 1.5


filtered_rows = process_csv_file(file_name, condition_column_index, condition_value, factor_column_index, factor)


# Display filtered rows
for row in filtered_rows:
    print(row)
```

53 Write a Python program to read a CSV file, reorder the columns to a new sequence, and display the first 5 rows of the modified DataFrame.

```
import pandas as pd


def reorder_columns(csv_file, new_sequence):
    # Read the CSV file into a DataFrame
    df = pd.read_csv(csv_file)


    # Reorder the columns
    df = df[new_sequence]
```

```
    # Display the first 5 rows of the modified DataFrame
    print("First 5 rows of the modified DataFrame:")
    print(df.head())


# Example usage:
csv_file = 'example.csv'  # Change this to the name of your CSV file
new_sequence = ['Column3', 'Column1', 'Column2']  # Change this to the new column sequence
reorder_columns(csv_file, new_sequence)
```

54 Create a program that reads a CSV file, calculates a new column by performing a mathematical operation on two existing columns, and appends the new column to the DataFrame.

```
import pandas as pd


def calculate_new_column(csv_file, column1, column2, new_column_name):
    # Read the CSV file into a DataFrame
    df = pd.read_csv(csv_file)


    # Perform the mathematical operation to calculate the new column
    df[new_column_name] = df[column1] + df[column2]
```

```
    # Display the first 5 rows of the DataFrame with the new column
    print("First 5 rows of the DataFrame with the new column:")
    print(df.head())


# Example usage:
csv_file = 'example.csv'  # Change this to the name of your CSV file
column1 = 'Column1'  # Change this to the name of the first existing column
column2 = 'Column2'  # Change this to the name of the second existing column
new_column_name = 'Sum'  # Change this to the name of the new column
calculate_new_column(csv_file, column1, column2, new_column_name)
```

55 Develop a program that reads data from a CSV file, removes rows with missing values in a specific column, and replaces missing values in another column with the mean of that column.

```
import pandas as pd


def process_csv_file(csv_file, column_with_missing_values, column_to_replace_missing_values):
```

```
    # Read the CSV file into a DataFrame
    df = pd.read_csv(csv_file)

    # Remove rows with missing values in the specified column
    df.dropna(subset=[column_with_missing_values], inplace=True)

    # Replace missing values in the specified column with the mean of that column
    mean_value = df[column_to_replace_missing_values].mean()
    df[column_to_replace_missing_values].fillna(mean_value, inplace=True)

    # Display the processed DataFrame
    print("Processed DataFrame:")
    print(df)

# Example usage:
csv_file = 'example.csv'  # Change this to the name of your CSV file
column_with_missing_values = 'Column2'  # Change this to the column with missing values
```

```
column_to_replace_missing_values = 'Column3'  # Change this to the column to replace missing values

process_csv_file(csv_file, column_with_missing_values, column_to_replace_missing_values)
```

56 Write a Python program that reads data from a CSV file using Pandas and creates a simple scatter plot using Matplotlib to visualize the relationship between two variables.

```
import pandas as pd

import matplotlib.pyplot as plt


def visualize_relationship(csv_file, x_column, y_column):
    # Read the CSV file into a DataFrame
    df = pd.read_csv(csv_file)


    # Extract the data for the scatter plot
    x_data = df[x_column]
    y_data = df[y_column]


    # Create the scatter plot
    plt.figure(figsize=(8, 6))
    plt.scatter(x_data, y_data, color='blue', alpha=0.5)
    plt.title('Scatter Plot of ' + y_column + ' vs ' + x_column)
    plt.xlabel(x_column)
    plt.ylabel(y_column)
```

```
    plt.grid(True)
    plt.show()


# Example usage:
csv_file = 'example.csv'  # Change this to the name of your CSV file
x_column = 'Column1'  # Change this to the name of the column for x-axis
y_column = 'Column2'  # Change this to the name of the column for y-axis
visualize_relationship(csv_file, x_column, y_column)
```

57 Create a program that reads categorical data from a CSV file using Pandas, and then generates a straightforward bar chart to show the counts of different categories.

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


def generate_bar_chart(csv_file, column):
    # Read the CSV file into a DataFrame
    df = pd.read_csv(csv_file)


    # Count the occurrences of each category in the specified column
    category_counts = df[column].value_counts()
```

```
    # Create the bar chart
    plt.figure(figsize=(8, 6))
    category_counts.plot(kind='bar', color='skyblue')
    plt.title('Bar Chart of Category Counts')
    plt.xlabel(column)
    plt.ylabel('Counts')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()


# Example usage:
csv_file = 'example.csv'  # Change this to the name of your CSV file
column = 'Category'  # Change this to the name of the categorical column
generate_bar_chart(csv_file, column)
```

58 Write a Python program that reads data from a CSV file using Pandas, performs data preprocessing or transformation with Numpy, and creates an informative scatter plot using Matplotlib to visualize multiple variables.

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

```
def preprocess_data(csv_file):
    # Read the CSV file into a DataFrame
    df = pd.read_csv(csv_file)


    # Perform data preprocessing or transformation
    # For example, let's scale the values of two columns using Min-Max scaling
    min_val = df[['Column1', 'Column2']].min()
    max_val = df[['Column1', 'Column2']].max()
    df[['Column1', 'Column2']] = (df[['Column1', 'Column2']] - min_val) / (max_val - min_val)


    return df


def create_scatter_plot(df):
    # Create the scatter plot
    plt.figure(figsize=(8, 6))
    plt.scatter(df['Column1'], df['Column2'], c=df['Column3'], cmap='viridis', alpha=0.8, s=100)
    plt.colorbar(label='Column3')
    plt.title('Scatter Plot of Column1 vs Column2')
    plt.xlabel('Column1')
    plt.ylabel('Column2')
```

```
    plt.grid(True)
    plt.show()


# Example usage:
csv_file = 'example.csv'  # Change this to the name of your CSV file


# Preprocess the data
processed_df = preprocess_data(csv_file)


# Create the scatter plot
create_scatter_plot(processed_df)
```

59. write a program to calcuate Greatest Common Divisor (GCD) of two numbers

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a


# Input two numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))


# Calculate the GCD
```

```
result = gcd(num1, num2)


# Output the result
print("The Greatest Common Divisor (GCD) of", num1, "and", num2, "is:", result)
```

60 Write a program to calculate Least Common Multiple (LCM) of two numbers

```
def gcd(a, b):
    # Calculate the Greatest Common Divisor (GCD) using Euclid's algorithm
    while b:
        a, b = b, a % b
    return a


def lcm(a, b):
    # Calculate the LCM using the formula: LCM(a, b) = (a * b) / GCD(a, b)
    return (a * b) // gcd(a, b)


# Input two numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))


# Calculate the LCM
```

```
result = lcm(num1, num2)


# Output the result
print("The Least Common Multiple (LCM) of", num1, "and", num2, "is:", result)
```

61 Python Program to Add Two Matrices

```
def add_matrices(matrix1, matrix2):
    # Check if the dimensions of the matrices are compatible for addition
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        print("Error: Matrices must have the same dimensions for addition")
        return None

    # Initialize an empty matrix to store the result
    result = []

    # Iterate over the rows of the matrices
    for i in range(len(matrix1)):
        row = []
        # Iterate over the columns of the matrices
        for j in range(len(matrix1[0])):
            # Add the corresponding elements from the two matrices
```

```
            row.append(matrix1[i][j] + matrix2[i][j])
        # Add the row to the result matrix
        result.append(row)

    return result

# Input two matrices from the user
def input_matrix():
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))
    matrix = []
    print("Enter the elements row-wise:")
    for i in range(rows):
        row = []
        for j in range(cols):
            row.append(int(input("Enter element at row {} and column {}: ".format(i+1, j+1))))
        matrix.append(row)
    return matrix

print("Enter the elements of the first matrix:")
matrix1 = input_matrix()
print("Enter the elements of the second matrix:")
```

```
matrix2 = input_matrix()

# Add the two matrices
result = add_matrices(matrix1, matrix2)

# Output the result
if result:
    print("The sum of the two matrices is:")
    for row in result:
        print(row)
```

62Write a program to find the hypotenuse of a right triangle

```
import math

def find_hypotenuse(a, b):
    # Calculate the square of each side
    a_squared = a ** 2
    b_squared = b ** 2

    # Calculate the sum of the squares
    sum_of_squares = a_squared + b_squared

    # Calculate the square root of the sum of the squares to find the hypotenuse
```

```
    hypotenuse = math.sqrt(sum_of_squares)

    return hypotenuse

# Input the lengths of the two shorter sides from the user
side1 = float(input("Enter the length of side 1: "))
side2 = float(input("Enter the length of side 2: "))

# Calculate the hypotenuse
hypotenuse = find_hypotenuse(side1, side2)

# Output the result
print("The length of the hypotenuse is:", hypotenuse)
```

63. Write a Python program to calculate the volume of a cube

```
def calculate_cube_volume(side):
    # Calculate the volume of the cube
    volume = side ** 3
    return volume

# Input the length of the side of the cube from the user
side_length = float(input("Enter the length of a side of the cube: "))

# Calculate the volume of the cube
```

```
cube_volume = calculate_cube_volume(side_length)


# Output the result
print("The volume of the cube with side length", side_length, "is:", cube_volume)
```

64 Write a python program to convert decimal to binary

```
def decimal_to_binary(decimal_num):
    binary_num = bin(decimal_num)[2:]
    return binary_num


# Input a decimal number from the user
decimal_number = int(input("Enter a decimal number: "))


# Convert the decimal number to binary
binary_number = decimal_to_binary(decimal_number)


# Output the result
print("Binary representation of", decimal_number, "is:", binary_number)
```

65, Write a python program to convert binary to decimal

```
def binary_to_decimal(binary_num):
    decimal_num = int(binary_num, 2)
    return decimal_num
```

```
# Input a binary number from the user
binary_number = input("Enter a binary number: ")

# Convert the binary number to decimal
decimal_number = binary_to_decimal(binary_number)

# Output the result
print("Decimal representation of", binary_number, "is:", decimal_number)
```

66 Write a program to find the average of a list of numbers in Python.

```
def calculate_average(numbers):
    if len(numbers) == 0:
        return 0
    total = sum(numbers)
    average = total / len(numbers)
    return average

# Input a list of numbers from the user
numbers = input("Enter a list of numbers separated by spaces: ").split()
numbers = [float(num) for num in numbers]
```

```
# Calculate the average of the numbers
average = calculate_average(numbers)

# Output the result
print("The average of the numbers is:", average)
```

67 Calculate the sum of all alternate even numbers from 1 until n in Python

```
def sum_alternate_even(n):
    # Initialize the sum
    total = 0

    # Iterate through numbers from 1 to n
    for i in range(1, n + 1):
        # Check if the number is even and its index is odd
        if i % 2 == 0 and i % 4 != 0:
            # Add the number to the sum
            total += i

    return total

# Input the value of n from the user
n = int(input("Enter the value of n: "))

# Calculate the sum of alternate even numbers
```

```
result = sum_alternate_even(n)


# Output the result
print("The sum of alternate even numbers from 1 to", n, "is:", result)
```

68 Calculate the sum of all alternate odd numbers from 1 until n in Python

```
def sum_alternate_odd(n):
    # Initialize the sum
    total = 0

    # Iterate through numbers from 1 to n
    for i in range(1, n + 1):
        # Check if the number is odd and its index is even
        if i % 2 != 0 and i % 4 == 0:
            # Add the number to the sum
            total += i

    return total

# Input the value of n from the user
n = int(input("Enter the value of n: "))

# Calculate the sum of alternate odd numbers
```

```
result = sum_alternate_odd(n)


# Output the result
print("The sum of alternate odd numbers from 1 to", n, "is:", result)
```

69 Write a python program to convert Celsius to Farenheit [°F = (°C × 9/5) + 32]

```
def celsius_to_fahrenheit(celsius):
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit


# Input temperature in Celsius from the user
celsius = float(input("Enter temperature in Celsius: "))


# Convert Celsius to Fahrenheit
fahrenheit = celsius_to_fahrenheit(celsius)


# Output the result
print("Temperature in Fahrenheit:", fahrenheit)
```

70 Write a python program to convert Farenheit to Celsius [°F = (°C × 9/5) + 32]

```
def fahrenheit_to_celsius(fahrenheit):
```

```
    celsius = (fahrenheit - 32) * 5/9
    return celsius

# Input temperature in Fahrenheit from the user
fahrenheit = float(input("Enter temperature in Fahrenheit: "))

# Convert Fahrenheit to Celsius
celsius = fahrenheit_to_celsius(fahrenheit)

# Output the result
print("Temperature in Celsius:", celsius)
```

71. Calculate the area of a parallelogram in Python.

```
def parallelogram_area(base, height):
    area = base * height
    return area

# Input the base and height of the parallelogram
base = float(input("Enter the length of the base of the parallelogram: "))
height = float(input("Enter the height of the parallelogram: "))

# Calculate the area of the parallelogram
area = parallelogram_area(base, height)
```

```
# Output the result
print("The area of the parallelogram is:", area)
```

72. Python Program to read a txt file, and print it as output without a newline

Python Program to read a txt file, and print it as output without a newline

```
def print_file_without_newline(file_name):
    with open(file_name, 'r') as file:
        for line in file:
            # Print each line without adding a newline
            print(line.strip(), end='')

# Input the name of the text file from the user
file_name = input("Enter the name of the text file: ")

# Print the contents of the text file without newline
print("Contents of the text file without newline:")
print_file_without_newline(file_name)
```

73. Calculate the volume of a rectangular prism in Python.

```
def rectangular_prism_volume(length, width, height):
    volume = length * width * height
    return volume

# Input the dimensions of the rectangular prism from the user
```

```
length = float(input("Enter the length of the rectangular prism: "))
width = float(input("Enter the width of the rectangular prism: "))
height = float(input("Enter the height of the rectangular prism: "))

# Calculate the volume of the rectangular prism
volume = rectangular_prism_volume(length, width, height)

# Output the result
print("The volume of the rectangular prism is:", volume)
```

74. Write a Python program to find the roots of a quadratic equation. [$x = (-b \pm \sqrt{(b^2 - 4ac)}) / (2a)$]

```
import cmath

def quadratic_roots(a, b, c):
    # Calculate the discriminant
    discriminant = b**2 - 4*a*c

    # Calculate the two roots using the quadratic formula
    root1 = (-b + cmath.sqrt(discriminant)) / (2*a)
    root2 = (-b - cmath.sqrt(discriminant)) / (2*a)
```

```
    return root1, root2

# Input the coefficients of the quadratic equation from the user
a = float(input("Enter the coefficient 'a': "))
b = float(input("Enter the coefficient 'b': "))
c = float(input("Enter the coefficient 'c': "))

# Calculate the roots of the quadratic equation
root1, root2 = quadratic_roots(a, b, c)

# Output the roots
print("Root 1:", root1)
print("Root 2:", root2)
```

75, Reverse a list without using built-in functions.

```
def reverse_list(lst):
    # Get the length of the list
    length = len(lst)
    # Iterate through the list up to the middle
    for i in range(length // 2):
        # Swap elements from the beginning and end of the list
        lst[i], lst[length - i - 1] = lst[length - i - 1], lst[i]

# Input a list of elements from the user
```

```
elements = input("Enter elements of the list separated by spaces: ").split()

# Convert elements to integers
elements = [int(element) for element in elements]

# Reverse the list
reverse_list(elements)

# Output the reversed list
print("Reversed list:", elements)
```

76. Write a Python function that checks if two strings are anagrams of each other. [Anagrams are words or phrases formed by rearranging the letters of a different word or phrase, using all the original letters exactly once.]

```
def are_anagrams(str1, str2):
    # Remove spaces and convert strings to lowercase
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()

    # Check if the length of both strings are the same
    if len(str1) != len(str2):
        return False

    # Count the occurrence of each character in both strings
```

```
    count1 = {}
    count2 = {}

    for char in str1:
        count1[char] = count1.get(char, 0) + 1

    for char in str2:
        count2[char] = count2.get(char, 0) + 1

    # Check if the dictionaries of character counts are equal
    return count1 == count2

# Example usage
str1 = "listen"
str2 = "silent"
if are_anagrams(str1, str2):
    print("The strings '{}' and '{}' are anagrams.".format(str1, str2))
else:
    print("The strings '{}' and '{}' are not anagrams.".format(str1, str2))
```

77. Find the longest word in a sentence.

```
def longest_word(sentence):
    # Split the sentence into words
```

```
    words = sentence.split()

    # Initialize variables to store the longest word and its length
    longest = ""
    max_length = 0

    # Iterate through the words to find the longest one
    for word in words:
        # Check if the current word is longer than the previous longest word
        if len(word) > max_length:
            longest = word
            max_length = len(word)

    return longest

# Example usage:
sentence = "The quick brown fox jumps over the lazy dog"
result = longest_word(sentence)
print("The longest word in the sentence is:", result)
```

78. sort a list using any one sorting technique

```
def main():
    # Define a list of unsorted elements
    unsorted_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
```

```
    # Sort the list using the sorted() function
    sorted_list = sorted(unsorted_list)

    # Print the sorted list
    print("Sorted list:", sorted_list)

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

79. Write a program to copy the contents of one text file to another in Python.

```
def copy_file(source_file, destination_file):
    try:
        # Open the source file for reading
        with open(source_file, 'r') as source:
            # Read the contents of the source file
            contents = source.read()

        # Open the destination file for writing
        with open(destination_file, 'w') as destination:
            # Write the contents to the destination file
            destination.write(contents)

        print("File copied successfully!")
```

```
    except FileNotFoundError:
        print("One or both files not found.")
    except Exception as e:
        print("An error occurred:", e)


# Example usage:
source_file = "source.txt"
destination_file = "destination.txt"
copy_file(source_file, destination_file)
```

80. program to count the occurrences of a specific character in a text file in Python.

```
def count_character_occurrences(file_name, target_character):
    try:
        # Open the file for reading
        with open(file_name, 'r') as file:
            # Initialize a counter for the occurrences
            count = 0
            # Read the file character by character
            for char in file.read():
                # Check if the character matches the target character
                if char == target_character:
                    count += 1
```

```
            return count

    except FileNotFoundError:
        print("File not found.")
        return None
    except Exception as e:
        print("An error occurred:", e)
        return None

# Example usage:
file_name = "example.txt"
target_character = 'a'
occurrences = count_character_occurrences(file_name,
target_character)
if occurrences is not None:
    print("Occurrences of '{}' in the file:
{}".format(target_character, occurrences))
```

81. Write a program that accepts a sentence and calculate the number of upper case letters and lower case letters.

```
def count_upper_lower(sentence):
    # Initialize counters for uppercase and lowercase letters
    upper_count = 0
    lower_count = 0
```

```
    # Iterate through each character in the sentence
    for char in sentence:
        # Check if the character is an uppercase letter
        if char.isupper():
            upper_count += 1
        # Check if the character is a lowercase letter
        elif char.islower():
            lower_count += 1

    return upper_count, lower_count

# Accept a sentence from the user
sentence = input("Enter a sentence: ")

# Calculate the number of uppercase and lowercase letters
upper, lower = count_upper_lower(sentence)

# Output the results
print("Number of uppercase letters:", upper)
print("Number of lowercase letters:", lower)
```

82. Define a function that can accept two strings as input and concatenate them and then stores it in a .txt file

```
def concatenate_strings_and_save(string1, string2,
file_name):
```

```
    # Concatenate the two strings
    concatenated_string = string1 + string2

    try:
        # Open the file for writing
        with open(file_name, 'w') as file:
            # Write the concatenated string to the file
            file.write(concatenated_string)

        print("Concatenated strings saved to '{}'
successfully!".format(file_name))

    except Exception as e:
        print("An error occurred:", e)

# Example usage:
string1 = "Hello, "
string2 = "world!"
file_name = "concatenated_strings.txt"
concatenate_strings_and_save(string1, string2, file_name)
```

83. Write a Python function that takes a year as input and determines whether it is a leap year. A year is a leap year if it is divisible by 4, except for years divisible by 100, but not divisible by 400.

```
def is_leap_year(year):
    """
    Determines if a given year is a leap year or not.

    Args:
    - year (int): The year to be checked.

    Returns:
    - bool: True if the year is a leap year, False otherwise.
    """
    if year % 4 == 0:  # If the year is divisible by 4
        if year % 100 == 0:  # If it's divisible by 100
            if year % 400 == 0:  # If it's divisible by 400
                return True
            else:
                return False
        else:
            return True
    else:
        return False

# Example usage:
year = 2024
```

```
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

84. Date difference calculator in python

```
from from
def km_to_miles(kilometers):
    """
    Convert kilometers to miles.

    Args:
    - kilometers (float): The distance in kilometers.

    Returns:
    - float: The distance converted to miles.
    """
    miles = kilometers / 1.60934
    return miles

# Example usage:
kilometers = float(input("Enter distance in kilometers: "))
miles = km_to_miles(kilometers)
print(f"{kilometers} kilometers is equal to {miles:.2f}
miles.")
```

86Python program to convert miles to kilometers [Miles = Kilometers / 1.60934]

```
def miles_to_km(miles):
    """
    Convert miles to kilometers.

    Args:
```

```
    - miles (float): The distance in miles.

    Returns:
    - float: The distance converted to kilometers.
    """
    kilometers = miles * 1.60934
    return kilometers

# Example usage:
miles = float(input("Enter distance in miles: "))
kilometers = miles_to_km(miles)
print(f"{miles} miles is equal to {kilometers:.2f}
kilometers.")
```

87 Calculate the area of a triangle in Python.

```
def area_of_triangle(a, b, c):
    """
    Calculate the area of a triangle using Heron's formula.

    Args:
    - a (float): Length of side a.
    - b (float): Length of side b.
    - c (float): Length of side c.

    Returns:
    - float: The area of the triangle.
    """
    # Calculate semi-perimeter
    s = (a + b + c) / 2

    # Calculate area using Heron's formula
    area = (s * (s - a) * (s - b) * (s - c)) ** 0.5
    return area
```

```
# Example usage:
a = 3
b = 4
c = 5
print("Area of the triangle:", area_of_triangle(a, b, c))
```

88 Python program to implement rock, paper, scissors

```
import random


def rock_paper_scissors():
    """
    Implements a simple rock, paper, scissors game.
    """
    choices = ['rock', 'paper', 'scissors']


    while True:
        # Get user choice
        user_choice = input("Enter your choice (rock, paper, scissors): ").lower()
        if user_choice not in choices:
            print("Invalid choice. Please enter 'rock', 'paper', or 'scissors'.")
            continue


        # Generate computer choice
        computer_choice = random.choice(choices)
```

```
        print("Computer chooses:", computer_choice)

        # Determine the winner
        if user_choice == computer_choice:
            print("It's a tie!")
        elif (user_choice == 'rock' and computer_choice == 'scissors') or \
             (user_choice == 'paper' and computer_choice == 'rock') or \
             (user_choice == 'scissors' and computer_choice == 'paper'):
            print("You win!")
        else:
            print("Computer wins!")

        # Ask if the user wants to play again
        play_again = input("Do you want to play again? (yes/no): ").lower()
        if play_again != 'yes':
            break

# Run the game
rock_paper_scissors()
```

89 Python program to implement an error handling concept

```
def divide_numbers(dividend, divisor):
    """
    Divide two numbers and handle division by zero error.

    Args:
    - dividend (float): The number to be divided.
    - divisor (float): The divisor.

    Returns:
    - float: The result of the division.
    """
    try:
        result = dividend / divisor
        return result
    except ZeroDivisionError:
        print("Error: Division by zero!")
        return None

# Example usage:
dividend = float(input("Enter the dividend: "))
divisor = float(input("Enter the divisor: "))

result = divide_numbers(dividend, divisor)
```

```
if result is not None:
    print(f"The result of the division is: {result}")
else:
    print("Division failed due to an error.")
```

90Write a Python program that creates all possible strings using the letters 'a', 'e', 'i', 'o', and 'I'. Ensure that each character is used only once.

```
from itertools import permutations


from itertools import permutations

def generate_strings():
    """
    Generate all possible strings using the letters 'a', 'e', 'i', 'o', and 'I'
    ensuring that each character is used only once.

    Returns:
    - list: A list of all possible strings.
    """
    letters = ['a', 'e', 'i', 'o', 'I']
    all_permutations = permutations(letters)
    all_strings = [''.join(permutation) for permutation in all_permutations]
    return all_strings

# Example usage:
possible_strings = generate_strings()
print("All possible strings:")
for string in possible_strings:
```

```
    print(string)
```

91Write a Python program to add two positive integers without using the '+' operator.

```
def add_without_plus(a, b):
    """
    Add two positive integers without using the '+' operator.

    Args:
    - a (int): The first positive integer.
    - b (int): The second positive integer.

    Returns:
    - int: The sum of the two integers.
    """
    while b != 0:
        carry = a & b  # Calculate the carry
        a = a ^ b      # Add without carrying
        b = carry << 1 # Shift the carry to the left by one bit
    return a

# Example usage:
num1 = int(input("Enter the first positive integer: "))
num2 = int(input("Enter the second positive integer: "))
result = add_without_plus(num1, num2)
```

```
print("The sum is:", result)
```

92 Write a Python program to find the mean,median,mode of a list of numbers.

```
from collections import Counter

from statistics import median


def calculate_mean(numbers):
    """
    Calculate the mean of a list of numbers.

    Args:
    - numbers (list): A list of numbers.

    Returns:
    - float: The mean of the numbers.
    """
    return sum(numbers) / len(numbers)


def calculate_median(numbers):
    """
    Calculate the median of a list of numbers.

    Args:
    - numbers (list): A list of numbers.
```

```
    Returns:
    - float: The median of the numbers.
    """
    return median(numbers)

def calculate_mode(numbers):
    """
    Calculate the mode of a list of numbers.

    Args:
    - numbers (list): A list of numbers.

    Returns:
    - list: A list of modes (in case of multiple modes).
    """
    counts = Counter(numbers)
    max_count = max(counts.values())
    modes = [num for num, count in counts.items() if count == max_count]
    return modes

# Example usage:
```

```
numbers = [1, 2, 3, 4, 5, 5, 6, 6, 6, 7]
print("Mean:", calculate_mean(numbers))
print("Median:", calculate_median(numbers))
print("Mode:", calculate_mode(numbers))
```

93. Python program to calculate simple interest

```
def calculate_simple_interest(principal, rate, time):
    """
    Calculate the simple interest.

    Args:
    - principal (float): The principal amount.
    - rate (float): The annual interest rate (as a percentage).
    - time (float): The time period (in years).

    Returns:
    - float: The simple interest.
    """
    interest = (principal * rate * time) / 100
    return interest

# Example usage:
principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the annual interest rate (as a percentage): "))
```

```
time = float(input("Enter the time period (in years): "))


simple_interest = calculate_simple_interest(principal, rate, time)
print("Simple Interest:", simple_interest)
```

94Python program to calculate compound interest

```
def calculate_compound_interest(principal, rate, time, n):
    """
    Calculate the compound interest.

    Args:
    - principal (float): The principal amount.
    - rate (float): The annual interest rate (as a percentage).
    - time (float): The time period (in years).
    - n (int): The number of times interest is compounded per time period.

    Returns:
    - float: The compound interest.
    """
    amount = principal * ((1 + (rate / (100 * n))) ** (n * time))
    compound_interest = amount - principal
    return compound_interest
```

```
# Example usage:
principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the annual interest rate (as a percentage): "))
time = float(input("Enter the time period (in years): "))
n = int(input("Enter the number of times interest is compounded per year: "))


compound_interest = calculate_compound_interest(principal, rate, time, n)
print("Compound Interest:", compound_interest)
```

95Write a Python program to read a string and replace a string "Python" with "Java" and "Java" with "Python" in a given string.

```
def replace_strings(input_string):
    """
    Replace "Python" with "Java" and "Java" with "Python" in a given string.

    Args:
    - input_string (str): The input string.

    Returns:
    - str: The modified string.
    """
```

```
    # Replace "Python" with a temporary placeholder
    temp_string = input_string.replace("Python",
"placeholder")
    # Replace "Java" with "Python"
    temp_string = temp_string.replace("Java", "Python")
    # Replace the temporary placeholder with "Java"
    output_string = temp_string.replace("placeholder", "Java")
    return output_string


# Example usage:
input_string = input("Enter a string: ")
modified_string = replace_strings(input_string)
print("Modified string:", modified_string)
```

96. Write a Python program to multiplication table of any given number

```
def multiplication_table(number):
    """
    Generate the multiplication table of a given number.


    Args:
    - number (int): The number whose multiplication table is to
be generated.


    Returns:
```

```
    - str: The multiplication table as a formatted string.
    """
    table = ""
    for i in range(1, 11):
        result = number * i
        table += f"{number} x {i} = {result}\n"
    return table

# Example usage:
num = int(input("Enter a number to generate its multiplication table: "))
print("Multiplication Table:")
print(multiplication_table(num))
```

97. Write a Python function to generate the Fibonacci series up to a specified number of terms.

```
def generate_fibonacci_series(num_terms):
    """
    Generate the Fibonacci series up to a specified number of terms.

    Args:
    - num_terms (int): The number of terms in the Fibonacci series to generate.
```

```
    Returns:
    - list: The Fibonacci series as a list.
    """
    fibonacci_series = []
    a, b = 0, 1
    for _ in range(num_terms):
        fibonacci_series.append(a)
        a, b = b, a + b
    return fibonacci_series

# Example usage:
num_terms = int(input("Enter the number of terms in the Fibonacci series: "))
fib_series = generate_fibonacci_series(num_terms)
print("Fibonacci series:")
print(fib_series)
```

98. Python Program for Array Rotation

```
def rotate_array(arr, d):
    """
    Rotate the elements of an array by d positions to the left.

    Args:
    - arr (list): The input array.
    - d (int): The number of positions to rotate the array.
```

```
    Returns:
    - list: The rotated array.
    """
    n = len(arr)
    d = d % n  # Adjusting for the case when d is greater than the length of the array
    return arr[d:] + arr[:d]

# Example usage:
arr = [1, 2, 3, 4, 5]
d = 2
rotated_arr = rotate_array(arr, d)
print("Original Array:", arr)
print(f"Array after rotating {d} positions to the left:", rotated_arr)
```

99Python Program to Swap Two Elements in a List

```
def rotate_array(arr, d):
    """
    Rotate the elements of an array by d positions to the left.

    Args:
    - arr (list): The input array.
    - d (int): The number of positions to rotate the array.
```

```
    Returns:
    - list: The rotated array.
    """
    n = len(arr)
    d = d % n  # Adjusting for the case when d is greater than the length of the array
    return arr[d:] + arr[:d]

# Example usage:
arr = [1, 2, 3, 4, 5]
d = 2
rotated_arr = rotate_array(arr, d)
print("Original Array:", arr)
print(f"Array after rotating {d} positions to the left:", rotated_arr)
```

100, Check if element exists in list in Python

```
def check_element_in_list(lst, element):
    """
    Check if an element exists in a list.

    Args:
    - lst (list): The input list.
    - element: The element to check for in the list.
```

```
    Returns:
    - bool: True if the element exists in the list, False otherwise.
    """
    return element in lst

# Example usage:
my_list = [1, 2, 3, 4, 5]
element_to_check = 3

if check_element_in_list(my_list, element_to_check):
    print(f"The element {element_to_check} exists in the list.")
else:
    print(f"The element {element_to_check} does not exist in the list.")
```

101. Write a program to calculate the square root of a given number.

```
def calculate_square_root(number):
    """
    Calculate the square root of a given number using the exponentiation operator.

    Args:
```

```
    - number (float): The number to calculate the square root of.

    Returns:
    - float: The square root of the number.
    """
    return number ** 0.5

# Example usage:
number = float(input("Enter a number to calculate its square root: "))
square_root = calculate_square_root(number)
print("Square root:", square_root)
```

102. Create a program that finds the maximum value between two given numbers.

```
def find_maximum(num1, num2):
    """
    Find the maximum value between two given numbers.

    Args:
    - num1 (float): The first number.
    - num2 (float): The second number.

    Returns:
```

```
    - float: The maximum of the two numbers.
    """
    return max(num1, num2)

# Example usage:
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
maximum_value = find_maximum(number1, number2)
print("Maximum value:", maximum_value)
```

103. Develop a program that calculates the sine of an angle in degrees.

```
import math

def calculate_sine(angle_degrees):
    """
    Calculate the sine of an angle in degrees.

    Args:
    - angle_degrees (float): The angle in degrees.

    Returns:
    - float: The sine of the angle.
    """
    # Convert angle from degrees to radians
```

```
    angle_radians = math.radians(angle_degrees)

    # Calculate sine of the angle
    sine_value = math.sin(angle_radians)

    return sine_value

# Example usage:
angle_degrees = float(input("Enter the angle in degrees: "))
sine_value = calculate_sine(angle_degrees)
print("Sine of the angle:", sine_value)
```

104. Write a program to find the floor value of a floating-point number.

```
import math

def find_floor_value(number):
    """
    Find the floor value of a floating-point number.

    Args:
    - number (float): The floating-point number.

    Returns:
    - int: The floor value of the number.
```

```
    """
    return math.floor(number)

# Example usage:
floating_point_number = float(input("Enter a floating-point number: "))
floor_value = find_floor_value(floating_point_number)
print("Floor value:", floor_value)
```

105. Develop a program to calculate the natural logarithm of a number.

```
import math

def calculate_natural_logarithm(number):
    """
    Calculate the natural logarithm of a number.

    Args:
    - number (float): The number.

    Returns:
    - float: The natural logarithm of the number.
    """
    return math.log(number)
```

```
# Example usage:
number = float(input("Enter a number: "))
natural_logarithm = calculate_natural_logarithm(number)
print("Natural logarithm:", natural_logarithm)
```

106. Write a program that generates a random number between a given range.

```
import random

def generate_random_number(start, end):
    """
    Generate a random number within a given range.

    Args:
    - start (int): The start of the range (inclusive).
    - end (int): The end of the range (inclusive).

    Returns:
    - int: The randomly generated number.
    """
    return random.randint(start, end)

# Example usage:
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
```

```
random_number = generate_random_number(start_range, end_range)
print("Random number:", random_number)
```

107Create a program to calculate the absolute value of a number.

```
def calculate_absolute_value(number):
    """
    Calculate the absolute value of a number.

    Args:
    - number (float): The number.

    Returns:
    - float: The absolute value of the number.
    """
    return abs(number)

# Example usage:
number = float(input("Enter a number: "))
absolute_value = calculate_absolute_value(number)
print("Absolute value:", absolute_value)
```

108Develop a program that calculates the cosine of an angle in degrees.

```
import math
```

```
def calculate_cosine(angle_degrees):
    """
    Calculate the cosine of an angle in degrees.

    Args:
    - angle_degrees (float): The angle in degrees.

    Returns:
    - float: The cosine of the angle.
    """
    # Convert angle from degrees to radians
    angle_radians = math.radians(angle_degrees)

    # Calculate cosine of the angle
    cosine_value = math.cos(angle_radians)

    return cosine_value

# Example usage:
angle_degrees = float(input("Enter the angle in degrees: "))
cosine_value = calculate_cosine(angle_degrees)
print("Cosine of the angle:", cosine_value)
```

109Write a program to round a floating-point number to the nearest integer.

```
def round_to_nearest_integer(number):
    """
    Round a floating-point number to the nearest integer.

    Args:
    - number (float): The floating-point number.

    Returns:
    - int: The rounded integer value.
    """
    return round(number)

# Example usage:
floating_point_number = float(input("Enter a floating-point number: "))
rounded_integer = round_to_nearest_integer(floating_point_number)
print("Rounded integer:", rounded_integer)
```

110 Create a program that calculates the power of a number.

```
def calculate_power(base, exponent):
    """
    Calculate the power of a number using the exponentiation operator.
```

```
    Args:
    - base (float): The base number.
    - exponent (float): The exponent.

    Returns:
    - float: The result of raising the base to the exponent power.
    """
    return base ** exponent

# Example usage:
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))
result = calculate_power(base, exponent)
print("Result:", result)
```

111 Develop a program to calculate the tangent of an angle in degrees.

```
import math

def calculate_tangent(angle_degrees):
    """
    Calculate the tangent of an angle in degrees.

    Args:
```

```
    - angle_degrees (float): The angle in degrees.

    Returns:
    - float: The tangent of the angle.
    """
    # Convert angle from degrees to radians
    angle_radians = math.radians(angle_degrees)

    # Calculate tangent of the angle
    tangent_value = math.tan(angle_radians)

    return tangent_value

# Example usage:
angle_degrees = float(input("Enter the angle in degrees: "))
tangent_value = calculate_tangent(angle_degrees)
print("Tangent of the angle:", tangent_value)
```

112 Develop a program to calculate the tangent of an angle in degrees.

Write a program to find the ceiling value of a floating-point number.

```
import math

def calculate_ceiling_value(number):
```

```
    """
    Calculate the ceiling value of a floating-point number.

    Args:
    - number (float): The floating-point number.

    Returns:
    - float: The ceiling value of the number.
    """
    return math.ceil(number)

# Example usage:
floating_point_number = float(input("Enter a floating-point number: "))
ceiling_value = calculate_ceiling_value(floating_point_number)
print("Ceiling value:", ceiling_value)
```

113Create a program that calculates the exponential value of a number.

```
import math

def calculate_exponential(number):
    """
    Calculate the exponential value of a number.
```

```
    Args:
    - number (float): The number.

    Returns:
    - float: The exponential value of the number.
    """
    return math.exp(number)

# Example usage:
number = float(input("Enter a number: "))
exponential_value = calculate_exponential(number)
print("Exponential value:", exponential_value)
```

114 Develop a program to calculate the hyperbolic sine of a number.

```
import math

def calculate_hyperbolic_sine(number):
    """
    Calculate the hyperbolic sine (sinh) of a number.

    Args:
    - number (float): The number.
```

```
    Returns:
    - float: The hyperbolic sine of the number.
    """
    return math.sinh(number)

# Example usage:
number = float(input("Enter a number: "))
hyperbolic_sine = calculate_hyperbolic_sine(number)
print("Hyperbolic sine:", hyperbolic_sine)
```

115Write a program to calculate the logarithm of a number with a given base.

```
import math

def calculate_logarithm(number, base):
    """
    Calculate the logarithm of a number with a given base.

    Args:
    - number (float): The number.
    - base (float): The base of the logarithm.

    Returns:
    - float: The logarithm of the number with the given base.
    """
```

```
    return math.log(number, base)


# Example usage:
number = float(input("Enter a number: "))
base = float(input("Enter the base of the logarithm: "))
logarithm_value = calculate_logarithm(number, base)
print(f"Logarithm of {number} with base {base}:",
logarithm_value)
```

116 Create a program that calculates the hyperbolic cosine of a number.

```
import math


def calculate_hyperbolic_cosh(number):
    """
    Calculate the hyperbolic cosine (cosh) of a number.

    Args:
    - number (float): The number.

    Returns:
    - float: The hyperbolic cosine of the number.
    """
    return math.cosh(number)
```

```
# Example usage:
number = float(input("Enter a number: "))
hyperbolic_cosh = calculate_hyperbolic_cosh(number)
print("Hyperbolic cosine:", hyperbolic_cosh)
```

117 Develop a program to calculate the hyperbolic tangent of a number.

```
import math

def calculate_hyperbolic_tanh(number):
    """
    Calculate the hyperbolic tangent (tanh) of a number.

    Args:
    - number (float): The number.

    Returns:
    - float: The hyperbolic tangent of the number.
    """
    return math.tanh(number)

# Example usage:
number = float(input("Enter a number: "))
hyperbolic_tanh = calculate_hyperbolic_tanh(number)
print("Hyperbolic tangent:", hyperbolic_tanh)
```

118 Write a program to calculate the arc sine of a number.

```
import math

def calculate_arc_sine(number):
    """
    Calculate the arc sine (arcsin) of a number.
```

```
    Args:
    - number (float): The number.

    Returns:
    - float: The arc sine of the number.
    """
    return math.asin(number)

# Example usage:
number = float(input("Enter a number: "))
arc_sine = calculate_arc_sine(number)
print("Arc sine:", arc_sine)
```

119 Create a program that calculates the arc cosine of a number.

```
import math

def calculate_arc_cosine(number):
    """
    Calculate the arc cosine (arccos) of a number.

    Args:
    - number (float): The number.

    Returns:
    - float: The arc cosine of the number.
    """
    return math.acos(number)

# Example usage:
number = float(input("Enter a number: "))
arc_cosine = calculate_arc_cosine(number)
print("Arc cosine:", arc_cosine)
```

120 Write a Python function that checks if a given string is a palindrome. [A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward as forward.]

```
def is_palindrome(string):
    """
    Check if a given string is a palindrome.

    Args:
    - string (str): The string to check.

    Returns:
    - bool: True if the string is a palindrome, False otherwise.
    """
    # Remove spaces and convert to lowercase
    string = string.replace(" ", "").lower()
    # Check if the string is equal to its reverse
    return string == string[::-1]

# Example usage:
input_string = input("Enter a string: ")
if is_palindrome(input_string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

121 Write a Python function to calculate the power of a number using recursion. The function should take two arguments: the base and the exponent.

```
def power(base, exponent):
    """
    Calculate the power of a number using recursion.

    Args:
```

```
    - base (int or float): The base number.
    - exponent (int): The exponent.

    Returns:
    - int or float: The result of raising the base to the exponent
power.
    """
    if exponent == 0:
        return 1
    elif exponent < 0:
        return 1 / power(base, -exponent)
    else:
        return base * power(base, exponent - 1)

# Example usage:
base = float(input("Enter the base number: "))
exponent = int(input("Enter the exponent: "))
result = power(base, exponent)
print(f"The result of {base} raised to the power of {exponent}
is:", result)
```

122 Write a Python program to check whether a number is a perfect number or not. [A perfect number is a positive integer that is equal to the sum of its positive divisors]

```
def is_perfect_number(number):
    """
    Check whether a number is a perfect number or not.

    Args:
    - number (int): The number to check.

    Returns:
    - bool: True if the number is a perfect number, False
otherwise.
```

```
    """
    if number <= 0:
        return False

    divisor_sum = 0
    for i in range(1, number):
        if number % i == 0:
            divisor_sum += i

    return divisor_sum == number

# Example usage:
num = int(input("Enter a number to check if it's a perfect number: "))
if is_perfect_number(num):
    print(f"{num} is a perfect number.")
else:
    print(f"{num} is not a perfect number.")
```

123 Write a Python program for creating a 2 dimensional matrix.

```
def create_matrix(rows, cols):
    """
    Create a 2-dimensional matrix.

    Args:
    - rows (int): The number of rows.
    - cols (int): The number of columns.

    Returns:
    - list of lists: The 2D matrix.
    """
    matrix = []
    for i in range(rows):
```

```
        row = []
        for j in range(cols):
            row.append(0)  # You can initialize elements to any
value
        matrix.append(row)
    return matrix

# Example usage:
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))
matrix = create_matrix(rows, cols)
print("Matrix:")
for row in matrix:
    print(row)
```

124 Write a Python program for matrix addition.

```
def matrix_addition(matrix1, matrix2):
    """
    Add two matrices element-wise.

    Args:
    - matrix1 (list of lists): The first matrix.
    - matrix2 (list of lists): The second matrix.

    Returns:
    - list of lists: The result of matrix addition.
    """
    if len(matrix1) != len(matrix2) or len(matrix1[0]) !=
len(matrix2[0]):
        raise ValueError("Matrices must have the same
dimensions for addition.")

    result = []
    for i in range(len(matrix1)):
```

```
        row = []
        for j in range(len(matrix1[0])):
            row.append(matrix1[i][j] + matrix2[i][j])
        result.append(row)
    return result

# Example usage:
matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix2 = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

print("Matrix 1:")
for row in matrix1:
    print(row)

print("\nMatrix 2:")
for row in matrix2:
    print(row)

try:
    result_matrix = matrix_addition(matrix1, matrix2)
    print("\nResult of Matrix Addition:")
    for row in result_matrix:
        print(row)
except ValueError as e:
    print("Error:", e)
```

125Creating an array nut not using numpy array or any standard compound data types of python.

```
def create_array(size):
    """
    Create an array with the specified size.

    Args:
    - size (int): The size of the array.
```

```
    Returns:
    - list: The created array.
    """
    return [0] * size

# Example usage:
array_size = int(input("Enter the size of the array: "))
my_array = create_array(array_size)
print("Array:", my_array)
```

126 Write a Python function to reverse a given string without using any built-in string reversal functions.

```
def reverse_string(input_string):
    """
    Reverse a given string without using built-in string reversal functions.

    Args:
    - input_string (str): The string to reverse.

    Returns:
    - str: The reversed string.
    """
    reversed_string = ""
    for i in range(len(input_string) - 1, -1, -1):
        reversed_string += input_string[i]
    return reversed_string

# Example usage:
input_str = input("Enter a string: ")
reversed_str = reverse_string(input_str)
print("Reversed string:", reversed_str)
```

127 Write a Python function that concatenates two strings without using the + operator.

```
def concatenate_strings(string1, string2):
    """
    Concatenate two strings without using the + operator.

    Args:
    - string1 (str): The first string.
    - string2 (str): The second string.

    Returns:
    - str: The concatenated string.
    """
    concatenated_string = ""
    for char in string1:
        concatenated_string += char
    for char in string2:
        concatenated_string += char
    return concatenated_string

# Example usage:
str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")
concatenated_str = concatenate_strings(str1, str2)
print("Concatenated string:", concatenated_str)
```

128Write a Python function that takes a sentence as input and capitalises the first letter of each word in the sentence.

```
def capitalize_first_letter(sentence):
    """
    Capitalize the first letter of each word in a sentence.

    Args:
```

```
    - sentence (str): The input sentence.

    Returns:
    - str: The sentence with the first letter of each word
capitalized.
    """
    words = sentence.split()  # Split the sentence into words
    capitalized_words = [word.capitalize() for word in words]
    return ' '.join(capitalized_words)

# Example usage:
input_sentence = input("Enter a sentence: ")
capitalized_sentence = capitalize_first_letter(input_sentence)
print("Capitalized sentence:", capitalized_sentence)
```

129Write a Python function that reverses the words in a sentence. For example, the input "Hello World" should be transformed into "World Hello".

```
def reverse_words(sentence):
    """
    Reverse the words in a sentence.

    Args:
    - sentence (str): The input sentence.

    Returns:
    - str: The sentence with the words reversed.
    """
    words = sentence.split()  # Split the sentence into words
    reversed_sentence = ' '.join(reversed(words))
    return reversed_sentence

# Example usage:
input_sentence = input("Enter a sentence: ")
```

```
reversed_sentence = reverse_words(input_sentence)
print("Reversed sentence:", reversed_sentence)
```

130Write a Python function that takes two strings as input and concatenates them to form a new string. For example, if the input strings are "Hello" and "World", the function should return "HelloWorld".

```
def concatenate_strings(str1, str2):
    """
    Concatenate two strings to form a new string.

    Args:
    - str1 (str): The first string.
    - str2 (str): The second string.

    Returns:
    - str: The concatenated string.
    """
    concatenated_string = ""
    for char in str1:
        concatenated_string += char
    for char in str2:
        concatenated_string += char
    return concatenated_string

# Example usage:
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")
concatenated_string = concatenate_strings(string1, string2)
print("Concatenated string:", concatenated_string)
```

131Create a Python program that acts as a simple calculator. It should take two numbers and an operator (+, -, *, /) as input

from the user and display the result of the corresponding operation.

```
def add(num1, num2):
    return num1 + num2

def subtract(num1, num2):
    return num1 - num2

def multiply(num1, num2):
    return num1 * num2

def divide(num1, num2):
    if num2 == 0:
        return "Error! Division by zero is not allowed."
    else:
        return num1 / num2

def calculator():
    print("Welcome to Simple Calculator!")
    print("Operations:")
    print("1. Addition (+)")
    print("2. Subtraction (-)")
    print("3. Multiplication (*)")
    print("4. Division (/)")

    operation = input("Enter the operation number (1/2/3/4): ")

    if operation not in ['1', '2', '3', '4']:
        print("Invalid operation number!")
        return

    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))
```

```
    if operation == '1':
        print("Result:", add(num1, num2))
    elif operation == '2':
        print("Result:", subtract(num1, num2))
    elif operation == '3':
        print("Result:", multiply(num1, num2))
    elif operation == '4':
        print("Result:", divide(num1, num2))

# Example usage:
calculator()
```

132Generate a numpy array with 100 random numbers between 0 and 1. Plot a histogram using seaborn to visualise the distribution of these numbers.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Generate an array with 100 random numbers between 0 and 1
random_numbers = np.random.rand(100)

# Plot histogram using Seaborn
sns.histplot(random_numbers, bins=10, kde=True)
plt.title("Distribution of Random Numbers")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```

133Write a Python program that takes a student's score as input and calculates the corresponding letter grade based on the following scale: A: 90-100 ; B: 80-89 ; C: 70-79 ; D: 60-69 ; F: Below 60

```
def calculate_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'F'

def main():
    try:
        score = float(input("Enter the student's score: "))
        if score < 0 or score > 100:
            print("Score must be between 0 and 100")
        else:
            grade = calculate_grade(score)
            print("The student's grade is:", grade)
    except ValueError:
        print("Invalid input. Please enter a numeric score.")

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

134 Create a 2D numpy array with 5 rows and 3 columns. Calculate the transpose of the array using numpy functions.

```
import numpy as np

# Create a 2D numpy array with 5 rows and 3 columns
array_2d = np.array([
    [1, 2, 3],
    [4, 5, 6],
```

```
    [7, 8, 9],
    [10, 11, 12],
    [13, 14, 15]
])

# Calculate the transpose of the array
transpose_array = np.transpose(array_2d)

print("Original 2D Array:")
print(array_2d)
print("\nTranspose of the Array:")
print(transpose_array)
```

135Given a numpy array, calculate the exponential of each element using numpy functions. Plot a scatter plot using seaborn to visualise the exponential values.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Generate random data for demonstration
np.random.seed(0)
data = np.random.rand(100)

# Calculate the exponential of each element
exponential_data = np.exp(data)

# Plot a scatter plot using Seaborn
sns.scatterplot(x=data, y=exponential_data)
plt.title('Exponential Values of Elements')
plt.xlabel('Original Values')
plt.ylabel('Exponential Values')
plt.show()
```

136Generate a sequence of 1000 random numbers between -10 and 10. Plot a boxplot using seaborn to visualise the distribution of the numbers.

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Generate a sequence of 1000 random numbers between -10 and 10
np.random.seed(0)
random_numbers = np.random.uniform(-10, 10, 1000)

# Plot a boxplot using Seaborn
sns.boxplot(data=random_numbers)
plt.title('Boxplot of Random Numbers')
plt.xlabel('Numbers')
plt.show()
```

137 Create a NumPy array of integers from 1 to 25. Print elements from index 5 to 15 using slicing.

```
import numpy as np

# Create a NumPy array of integers from 1 to 25
array = np.arange(1, 26)

# Print elements from index 5 to 15 using slicing
print("Elements from index 5 to 15:", array[5:16])
```

138 Create two NumPy arrays of the same shape. Concatenate them along axis 0 (rows) and axis 1 (columns). Print the concatenated arrays.

```
import numpy as np

# Create two NumPy arrays of the same shape
```

```
array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])

array2 = np.array([[7, 8, 9],
                   [10, 11, 12]])

# Concatenate along axis 0 (rows)
concatenated_rows = np.concatenate((array1, array2), axis=0)

# Concatenate along axis 1 (columns)
concatenated_columns = np.concatenate((array1, array2),
axis=1)

print("Concatenated along axis 0 (rows):")
print(concatenated_rows)

print("\nConcatenated along axis 1 (columns):")
print(concatenated_columns)
```

139 Create two NumPy arrays of the same shape. Perform an element-wise comparison (e.g., greater than, less than) between the arrays and print the resulting Boolean

```
import numpy as np

# Create two NumPy arrays of the same shape
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([5, 4, 3, 2, 1])

# Perform element-wise comparison (e.g., greater than, less
than)
greater_than_result = array1 > array2
less_than_result = array1 < array2

# Print the resulting Boolean arrays
```

```
print("Array 1:", array1)
print("Array 2:", array2)
print("Array 1 > Array 2:", greater_than_result)
print("Array 1 < Array 2:", less_than_result)
```

140 Create a NumPy array with dimensions (2, 3, 4). Flatten the array and print the flattened array. [A flattened array is an array that has been transformed from a multi-dimensional structure into a one-dimensional sequence]

```
import numpy as np

# Create a NumPy array with dimensions (2, 3, 4)
array = np.array([[[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]],
                  [[13, 14, 15, 16],
                   [17, 18, 19, 20],
                   [21, 22, 23, 24]]])

# Flatten the array
flattened_array = array.flatten()

# Print the flattened array
print("Original Array:")
print(array)
print("\nFlattened Array:")
print(flattened_array)
```