MLE PRACTICAL QUESTIONS

1.Create a NumPy array from a Python list.

```
Import numpy as np
python_list = [1, 2, 3, 4, 5]
numpy_array = np.array(python_list)
print("1. NumPy Array:", numpy_array)
```

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

```
>>> import numpy as np
>>> python_list = [1,2,3,4,5]
>>> numpy_array = np.array(python_list)
>>> print("Shape of NumPy array:", numpy_array.shape)
```

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

```
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
element_wise_addition = array1 + array2
```

```
print("3. Element-wise addition:", element_wise_addition)
```

4.Calculate the mean of a NumPy array.

```
array = np.array([1, 2, 3, 4, 5])
mean = np.mean(array)
print("Mean of NumPy array:", mean)
```

```
# 5. Calculate the median of a NumPy array.
median = np.median(array)
print("Median of NumPy array:", median)
```

```
# 6. How to find the maximum and minimum values in a NumPy array?
maximum = np.max(array)
minimum = np.min(array)
print("Maximum value:", maximum)
print("Minimum value:", minimum)
```

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

```
import numpy as np


array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6]])


horizontal_concatenation = np.concatenate((array1, array2), axis=1)
vertical_concatenation = np.concatenate((array1, array2), axis=0)


print("7. Horizontal Concatenation:\n", horizontal_concatenation)
print("   Vertical Concatenation:\n", vertical_concatenation)


# 8. Calculate the dot product of two NumPy arrays.
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6], [7, 8]])
dot_product = np.dot(array1, array2)
print("Dot Product:\n", dot_product)


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

```
array = np.array([1, 2, 3, 1, 2, 1, 3, 3, 4])
unique_elements, counts = np.unique(array,
return_counts=True)
print("9. Unique elements:", unique_elements)
print("    Counts:", counts)
```

op:

```
Unique elements: [1 2 3 4]
    Counts: [3 2 3 1]
```

```
# 10. Create a NumPy array with elements 1 to 10.
array_1_to_10 = np.arange(1, 11)
print("10. Array with elements 1 to 10:", array_1_to_10)

# 11. Create a 3x3 identity matrix using NumPy
identity_matrix = np.eye(3)
print("11. 3x3 Identity Matrix:\n", identity_matrix)

# 12. Create a NumPy array with a specified upper and lower
limit.
lower_limit = 5
upper_limit = 15
```

```
array_limits = np.arange(lower_limit, upper_limit+1)
print("12. Array with lower and upper limits:", array_limits)

# 13. Calculate the sum of all elements in a NumPy array.
array = np.array([1, 2, 3, 4, 5])
sum_array = np.sum(array)
print("13. Sum of all elements:", sum_array)

# 14. Replace all even numbers in a NumPy array with 0.
array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
array[array % 2 == 0] = 0
print("14. Array with even numbers replaced by 0:", array)

# 15. Convert a NumPy array to a Python list.
array = np.array([1, 2, 3, 4, 5])
python_list = array.tolist()
print("15. Converted Python list:", python_list)

# 16. Calculate the inverse of a square NumPy matrix.
matrix = np.array([[1, 2], [3, 4]])
inverse_matrix = np.linalg.inv(matrix)
```

```
print("16. Inverse of the matrix:\n", inverse_matrix)

# 17. Remove all NaN values from a NumPy array.
array_with_nan = np.array([1, np.nan, 2, 3, np.nan, 4])
array_without_nan =
array_with_nan[~np.isnan(array_with_nan)]
print("17. Array without NaN values:", array_without_nan)

# 18. Perform element-wise subtraction of two NumPy arrays.
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
element_wise_subtraction = array1 - array2
print("18. Element-wise subtraction:",
element_wise_subtraction)

# 19. Perform element-wise division of two NumPy arrays.
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
element_wise_division = array1 / array2
print("19. Element-wise division:", element_wise_division)
```

```
# 20. Find the indices of the minimum and maximum values in a NumPy array.
array = np.array([1, 3, 5, 2, 4])
min_index = np.argmin(array)
max_index = np.argmax(array)
print("20. Index of minimum value:", min_index)
print("    Index of maximum value:", max_index)

# 21. Check if two NumPy arrays are equal.
array1 = np.array([1, 2, 3])
array2 = np.array([1, 2, 3])
arrays_equal = np.array_equal(array1, array2)
print("21. Arrays are equal:", arrays_equal)

# 22. Extract specific rows and columns from a NumPy array.
array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
rows = [0, 2]
cols = [0, 2]
extracted_elements = array[rows][:, cols]
print("22. Extracted elements:\n", extracted_elements)
```

```
# 23. Sort a NumPy array in ascending order.
array = np.array([3, 1, 2, 5, 4])
sorted_array = np.sort(array)
print("23. Sorted array in ascending order:", sorted_array)

# 24. Sort a NumPy array in descending order.
array = np.array([3, 1, 2, 5, 4])
sorted_array_desc = np.sort(array)[::-1]
print("24. Sorted array in descending order:",
sorted_array_desc)

# 25. Round the elements of a NumPy array to the nearest
integer.
array = np.array([1.1, 2.5, 3.7])
rounded_array = np.round(array)
print("25. Rounded array:", rounded_array)

# 26. Check if any element in a NumPy array is NaN.
array = np.array([1, np.nan, 2, 3])
contains_nan = np.isnan(array).any()
print("26. Array contains NaN:", contains_nan)
```

```
# 27. Create a NumPy array and print its size, and data type.
array = np.array([1, 2, 3, 4, 5])
print("27. Size of array:", array.size)
print("    Data type of array:", array.dtype)

# 28. Write a Python program to print a pyramid pattern
def pyramid_pattern(n):
    for i in range(n):
        print(" "*(n-i-1) + "*"*(2*i+1))

print("28. Pyramid Pattern:")
pyramid_pattern(5)

# 29. Create a Python program to print a diamond pattern with a given number of rows.
def diamond_pattern(n):
    for i in range(n):
        print(" "*(n-i-1) + "*"*(2*i+1))
    for i in range(n-2, -1, -1):
        print(" "*(n-i-1) + "*"*(2*i+1))
```

```
print("29. Diamond Pattern:")
diamond_pattern(5)


# 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 number_pyramid(n):
    for i in range(1, n+1):
        print(" "*(n-i) + " ".join(str(j) for j in range(1, i*2)))

print("30. Pyramid Pattern with Numbers:")
number_pyramid(5)


# 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(num):
    original_num_square = num ** 2
    reversed_num = int(str(num)[::-1])
    reversed_num_square = reversed_num ** 2
```

```
    return original_num_square ==
int(str(reversed_num_square)[::-1])


num = 121
if is_adam_number(num):
    print("31. {} is an Adam number.".format(num))
else:
    print("31. {} is not an Adam number.".format(num))


# 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(num):
    square = num ** 2
    return str(square).endswith(str(num))


num = 25
if is_automorphic_number(num):
    print("32. {} is an automorphic number.".format(num))
else:
    print("32. {} is not an automorphic number.".format(num))
```

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

```
def is_perfect_number(num):
    divisors_sum = sum([i for i in range(1, num) if num % i == 0])
    return divisors_sum == num


lower_limit = 1
upper_limit = 100
perfect_numbers = [num for num in range(lower_limit, upper_limit+1) if is_perfect_number(num)]
print("33. Perfect numbers between {} and {}: {}".format(lower_limit, upper_limit, perfect_numbers))
```

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(num):
    seen = set()
    while num != 1 and num not in seen:
        seen.add(num)
```

```
        num = sum(int(digit)**2 for digit in str(num))
    return num == 1


num = 19
if is_happy_number(num):
    print("34. {} is a happy number.".format(num))
else:
    print("34. {} is not a happy number.".format(num))


# 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)
def is_armstrong_number(num):
    num_str = str(num)
    power = len(num_str)
    sum_of_powers = sum(int(digit)**power for digit in num_str)
    return sum_of_powers == num


num = 153
if is_armstrong_number(num):
```

```
    print("35. {} is an Armstrong number.".format(num))
else:
    print("35. {} is not an Armstrong number.".format(num))


# 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, emp_id):
        self.name = name
        self.emp_id = emp_id

    @abstractmethod
    def calculate_salary(self):
        pass

    @abstractmethod
    def display_information(self):
```

```
        pass

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

    def calculate_salary(self):
        return self.salary

    def display_information(self):
        print("Manager Information:")
        print("Name:", self.name)
        print("Employee ID:", self.emp_id)
        print("Salary:", self.salary)

class Developer(Employee):
    def __init__(self, name, emp_id, hours_worked,
hourly_rate):
        super().__init__(name, emp_id)
        self.hours_worked = hours_worked
```

```
        self.hourly_rate = hourly_rate

    def calculate_salary(self):
        return self.hours_worked * self.hourly_rate

    def display_information(self):
        print("Developer Information:")
        print("Name:", self.name)
        print("Employee ID:", self.emp_id)
        print("Hours Worked:", self.hours_worked)
        print("Hourly Rate:", self.hourly_rate)

class Designer(Employee):
    def __init__(self, name, emp_id, projects_completed,
project_bonus):
        super().__init__(name, emp_id)
        self.projects_completed = projects_completed
        self.project_bonus = project_bonus

    def calculate_salary(self):
        return self.projects_completed * self.project_bonus
```

```
    def display_information(self):
        print("Designer Information:")
        print("Name:", self.name)
        print("Employee ID:", self.emp_id)
        print("Projects Completed:", self.projects_completed)
        print("Project Bonus:", self.project_bonus)

# Example usage
manager = Manager("John Doe", 1001, 60000)
developer = Developer("Jane Smith", 2001, 160, 50)
designer = Designer("Alice Johnson", 3001, 10, 1000)

print("--- Employee Information ---")
manager.display_information()
print("Calculated Salary:", manager.calculate_salary())
print()

developer.display_information()
print("Calculated Salary:", developer.calculate_salary())
print()
```

```
designer.display_information()
print("Calculated Salary:", designer.calculate_salary())
print()


# 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.
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=0.03):
        super().__init__(account_number, balance)
        self.interest_rate = interest_rate

    def deposit(self, amount):
        self.balance += amount

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

    def add_interest(self):
        self.balance += self.balance * self.interest_rate

class CheckingAccount(BankAccount):
```

```
    def __init__(self, account_number, balance=0,
overdraft_fee=25):
        super().__init__(account_number, balance)
        self.overdraft_fee = overdraft_fee

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            self.balance -= self.overdraft_fee
            print("Overdraft fee charged.")

# Example usage
savings_account = SavingsAccount("SA12345", 1000)
checking_account = CheckingAccount("CA54321", 500)

print("--- Before Transactions ---")
print("Savings Account Balance:", savings_account.balance)
```

```
print("Checking Account Balance:",
checking_account.balance)


savings_account.deposit(500)
checking_account.withdraw(200)


print("--- After Transactions ---")
print("Savings Account Balance:", savings_account.balance)
print("Checking Account Balance:",
checking_account.balance)


# 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.
class TextProcessor(ABC):
    @abstractmethod
    def format_text(self, text):
        pass


    @abstractmethod
```

```
    def analyze_text(self, text):
        pass

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

    def analyze_text(self, text):
        return {
            "length": len(text),
            "word_count": len(text.split()),
            "character_count": sum(1 for char in text if
char.isalpha())
        }

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

    def analyze_text(self, text):
        return {
```

```
            "length": len(text),
            "word_count": len(text.split()),
            "character_count": sum(1 for char in text if
char.isalpha())
        }

# Example usage
text = "This is a Sample Text for Testing"
upper_formatter = UpperCaseFormatter()
lower_formatter = LowerCaseFormatter()

print("--- Uppercase Formatting ---")
formatted_text_upper = upper_formatter.format_text(text)
print("Formatted Text:", formatted_text_upper)
print("Text Analysis:", upper_formatter.analyze_text(text))

print("\n--- Lowercase Formatting ---")
formatted_text_lower = lower_formatter.format_text(text)
print("Formatted Text:", formatted_text_lower)
print("Text Analysis:", lower_formatter.analyze_text(text))
```

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

from math import pi


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 pi * self.radius ** 2

    def calculate_perimeter(self):
        return 2 * 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("\n--- Rectangle ---")
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(ABC):
    def __init__(self, name, price):
        self.name = name
        self.price = price

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

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


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


    def calculate_shipping_cost(self):
        return 0.05 * self.weight  # Assuming shipping cost is
5% of weight


    def product_details(self):
        return f"Name: {self.name}, Price: ${self.price}, Weight:
{self.weight} kg"


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

```
    def calculate_shipping_cost(self):
        return 5  # Flat shipping rate for clothing

    def product_details(self):
        return f"Name: {self.name}, Price: ${self.price}, Size: {self.size}"

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

    def calculate_shipping_cost(self):
        return 3  # Flat shipping rate for books

    def product_details(self):
        return f"Name: {self.name}, Price: ${self.price}, Author: {self.author}"

# Example usage
```

```
laptop = Electronics("Laptop", 1000, 2)
shirt = Clothing("T-Shirt", 20, "M")
book = Books("Python Programming", 50, "Guido van Rossum")

print("--- Product Details ---")
print(laptop.product_details())
print("Shipping Cost:", laptop.calculate_shipping_cost())

print(shirt.product_details())
print("Shipping Cost:", shirt.calculate_shipping_cost())

print(book.product_details())
print("Shipping Cost:", book.calculate_shipping_cost())



# 41. Write a Python program that calculates the factorial of a given number using recursion.
def factorial(n):
    if n == 0:
        return 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_without_recursion(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_without_recursion(number))
```

43. Write a Python function that takes an integer as input and determines whether it is a prime number or not.

```
def is_prime(n):
    if n <= 1:
        return False
    elif n == 2:
        return True
    elif n % 2 == 0:
        return False
    else:
        for i in range(3, int(n**0.5) + 1, 2):
            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():
    target_number = random.randint(1, 100)
    attempts = 0
    max_attempts = 5


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


    while attempts < max_attempts:
        attempts += 1
        guess = int(input("Enter your guess: "))


        if guess < target_number:
            print("Too low! Try again.")
```

```
        elif guess > target_number:
            print("Too high! Try again.")
        else:
            print(f"Congratulations! You guessed the number {target_number} correctly in {attempts} attempts.")
            break
    else:
        print(f"Sorry, you've used all {max_attempts} attempts. The correct number was {target_number}.")

# Example usage
number_guessing_game()
```

Certainly! Let's continue:

```python
# 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):
    primes = []
    for num in range(start, end + 1):
```

```
        if num > 1:
            for i in range(2, int(num**0.5) + 1):
                if (num % i) == 0:
                    break
            else:
                primes.append(num)
    return primes


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


# 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:
        if char.isalpha():
```

```
            shifted_char = chr((ord(char.lower()) - 97 + shift) %
26 + 97)
            encrypted_text += shifted_char.upper() if
char.isupper() else shifted_char
        else:
            encrypted_text += char
    return encrypted_text

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

# 47. Write a program that performs operations on a list, such
as adding elements, removing duplicates, and sorting.
def list_operations(lst):
    # Adding elements
    lst.append(6)
```

```
    # Removing duplicates
    lst = list(set(lst))

    # Sorting
    lst.sort()

    return lst

# Example usage
original_list = [3, 1, 2, 3, 5, 4, 2]
print("Original list:", original_list)
modified_list = list_operations(original_list)
print("Modified list:", modified_list)

# 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):
    words = sentence.split()
    return len(words)
```

```
# Example usage
user_sentence = input("Enter a sentence: ")
word_count = count_words(user_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 separate_even_odd_numbers(lst):
    even_numbers = [num for num in lst if num % 2 == 0]
    odd_numbers = [num for num in lst if num % 2 != 0]
    return even_numbers, odd_numbers

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

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

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

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

# Indexing
print("Element at index 1:", concatenated_tuple[1])

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

# 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 read_csv_file(filename, increment_amount):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
```

```
        for row in reader:
            second_column_value = row[1]
            third_column_value = int(row[2]) +
increment_amount
            print("Second column value:", second_column_value)
            print("Incremented third column value:",
third_column_value)


# Example usage
csv_file = "data.csv"
increment_amount = 10
read_csv_file(csv_file, 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).
def filter_and_multiply_csv(filename,
condition_column_index, condition_value,
multiply_column_index, multiply_factor):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
```

```
            if float(row[condition_column_index]) >
condition_value:
                multiplied_value =
float(row[multiply_column_index]) * multiply_factor
                print("Filtered row:", row)
                print("Multiplied value:", multiplied_value)


# Example usage
csv_file = "data.csv"
condition_column_index = 1
condition_value = 50
multiply_column_index = 2
multiply_factor = 1.5
filter_and_multiply_csv(csv_file, condition_column_index,
condition_value, multiply_column_index, multiply_factor)


# 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_csv_columns(filename, new_column_order):
```

```
    df = pd.read_csv(filename)
    reordered_df = df[new_column_order]
    print("First 5 rows of modified DataFrame:")
    print(reordered_df.head())

# Example usage
csv_file = "data.csv"
new_column_order = ['Column2', 'Column1', 'Column3']  # Specify new column order
reorder_csv_columns(csv_file, new_column_order)

# 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.
def calculate_and_append_column(filename, column1_name, column2_name, new_column_name):
    df = pd.read_csv(filename)
    df[new_column_name] = df[column1_name] * df[column2_name]
    print("DataFrame with the new calculated column:")
    print(df)
```

```
# Example usage
csv_file = "data.csv"
column1_name = 'Column1'
column2_name = 'Column2'
new_column_name = 'NewColumn'
calculate_and_append_column(csv_file, column1_name, column2_name, 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.
def handle_missing_values(filename, column_with_missing_values, column_to_replace_with_mean):
    df = pd.read_csv(filename)
    # Remove rows with missing values in a specific column
    df.dropna(subset=[column_with_missing_values], inplace=True)
    # Replace missing values in another column with the mean of that column
```

```
df[column_to_replace_with_mean].fillna(df[column_to_replace_with_mean].mean(), inplace=True)
    print("DataFrame after handling missing values:")
    print(df)

# Example usage
csv

_file = "data.csv"
column_with_missing_values = 'Column1'
column_to_replace_with_mean = 'Column2'
handle_missing_values(csv_file,
column_with_missing_values,
column_to_replace_with_mean)

# 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 matplotlib.pyplot as plt

def create_scatter_plot(filename, x_column, y_column):
```

```
    df = pd.read_csv(filename)
    plt.scatter(df[x_column], df[y_column])
    plt.xlabel(x_column)
    plt.ylabel(y_column)
    plt.title("Scatter Plot")
    plt.show()

# Example usage
csv_file = "data.csv"
x_column = 'Column1'
y_column = 'Column2'
create_scatter_plot(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.
def create_bar_chart(filename, category_column):
    df = pd.read_csv(filename)
    category_counts = df[category_column].value_counts()
    category_counts.plot(kind='bar')
    plt.xlabel(category_column)
```

```
    plt.ylabel("Counts")
    plt.title("Bar Chart")
    plt.show()


# Example usage
csv_file = "data.csv"
category_column = 'Category'
create_bar_chart(csv_file, category_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.
def preprocess_and_create_scatter_plot(filename, columns_to_transform):
    df = pd.read_csv(filename)
    # Perform data preprocessing or transformation with NumPy
    for column in columns_to_transform:
        df[column] = np.log(df[column])  # Example transformation (logarithm)
    # Create scatter plot
```

```
    plt.scatter(df[columns_to_transform[0]],
df[columns_to_transform[1]])
    plt.xlabel(columns_to_transform[0])
    plt.ylabel(columns_to_transform[1])
    plt.title("Scatter Plot with Transformed Variables")
    plt.show()

# Example usage
csv_file = "data.csv"
columns_to_transform = ['Column1', 'Column2']
preprocess_and_create_scatter_plot(csv_file,
columns_to_transform)

# 59. Write a program to calculate Greatest Common Divisor
(GCD) of two numbers.
import math

def calculate_gcd(a, b):
    return math.gcd(a, b)

# Example usage
```

```
num1 = 12

num2 = 18

print(f"GCD of {num1} and {num2}:", calculate_gcd(num1, num2))


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

def calculate_lcm(a, b):

    return abs(a*b) // math.gcd(a, b)


# Example usage

num1 = 12

num2 = 18

print(f"LCM of {num1} and {num2}:", calculate_lcm(num1, num2))


# 61. Python Program to Add Two Matrices

def add_matrices(matrix1, matrix2):

    result = [[0]*len(matrix1[0]) for _ in range(len(matrix1))]

    for i in range(len(matrix1)):

        for j in range(len(matrix1[0])):
```

```
            result[i][j] = matrix1[i][j] + matrix2[i][j]
    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("Sum of the matrices:")
for row in add_matrices(matrix1, matrix2):
    print(row)

# 62. Write a program to find the hypotenuse of a right triangle
def find_hypotenuse(base, height):
    return math.sqrt(base**2 + height**2)

# Example usage
base = 3
height = 4
print("Hypotenuse of the right triangle:", find_hypotenuse(base, height))
```

```
# 63. Write a Python program to calculate the volume of a cube
def cube_volume(side):
    return side**3

# Example usage
side_length = 5
print("Volume of the cube:", cube_volume(side_length))

# 64. Write a python program to convert decimal to binary
def decimal_to_binary(decimal):
    return bin(decimal).replace("0b", "")

# Example usage
decimal_num = 10
print(f"Binary representation of {decimal_num}:", decimal_to_binary(decimal_num))

# 65. Write a python program to convert binary to decimal
def binary_to_decimal(binary):
    return int(binary, 2)
```

```
# Example usage
binary_num = "1010"
print(f"Decimal representation of {binary_num}:", binary_to_decimal(binary_num))


# 66. Write a program to find the average of a list of numbers in Python.
def calculate_average(numbers):
    return sum(numbers) / len(numbers)

# Example usage
numbers_list = [1, 2, 3, 4, 5]
print("Average of the numbers:", calculate_average(numbers_list))


# 67. Calculate the sum of all alternate even numbers from 1 until n in Python
def sum_alternate_even_numbers(n):
    return sum([i for i in range(2, n + 1, 4)])

# Example usage
```

```
n = 10
print("Sum of all alternate even numbers from 1 until", n, ":", sum_alternate_even_numbers(n))


# 68. Calculate the sum of all alternate odd numbers from 1 until n in Python
def sum_alternate_odd_numbers(n):
    return sum([i for i in range(1, n + 1, 4)])


# Example usage
n = 10
print("Sum of all alternate odd numbers from 1 until", n, ":", sum_alternate_odd_numbers(n))


# 69. Write a python program to convert Celsius to Fahrenheit [°F = (°C × 9/5) + 32]
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32


# Example usage
celsius_temp = 25
```

```
print(f"{celsius_temp}°C is equal to
{celsius_to_fahrenheit(celsius_temp)}°F")


# 70. Write a python program to convert Fahrenheit to Celsius
[°C = (°F - 32) * 5/9]

def fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9


# Example usage

fahrenheit_temp = 77

print(f"{fahrenheit_temp}°F is equal to
{fahrenheit_to_celsius(fahrenheit_temp)}°C")


# 71. Calculate the area of a parallelogram in Python.

def parallelogram_area(base, height):
    return base * height


# Example usage

base = 5

height = 8

print("Area of the parallelogram:", parallelogram_area(base,
height))
```

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

```
def read_file_without_newline(filename):
    with open(filename, 'r') as file:
        for line in file:
            print(line.rstrip(), end="")

# Example usage
text_file = "sample.txt"
print("Content of the file without newlines:")
read_file_without_newline(text_file)
```

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

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

# Example usage
length = 4
```

```
width = 3
height = 5
print("Volume of the rectangular prism:", rectangular_prism_volume(length, width, height))


# 74. Write a Python program to find the roots of a quadratic equation. [x = (-b ± √(b² - 4ac)) / (2a)]
def quadratic_roots(a, b, c):
    discriminant = b**2 - 4*a*c
    if discriminant > 0:
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        return root1, root2
    elif discriminant == 0:
        root = -b / (2*a)
        return root, root
    else:
        real_part = -b / (2*a)
        imaginary_part = math.sqrt(abs(discriminant)) / (2*a)
        return (real_part, imaginary_part), (real_part, -imaginary_part)
```

```
# Example usage
a = 1
b = -3
c = 2
print("Roots of the quadratic equation:", quadratic_roots(a, b, c))


# 75. Reverse a list without using built-in functions.
def reverse_list(lst):
    reversed_list = []
    for i in range(len(lst)-1, -1, -1):
        reversed_list.append(lst[i])
    return reversed_list

# Example usage
original_list = [1, 2, 3, 4, 5]
print("Original list:", original_list)
print("Reversed list:", reverse_list(original_list))
```

76. Write a Python function that checks if two strings are anagrams of each other.

```
def are_anagrams(str1, str2):
    return sorted(str1) == sorted(str2)


# Example usage
string1 = "listen"
string2 = "silent"
print(f"Are '{string1}' and '{string2}' anagrams?", are_anagrams(string1, string2))
```

77. Find the longest word in a sentence.

```
def find_longest_word(sentence):
    words = sentence.split()
    return max(words, key=len)


# Example usage
user_sentence = "This is a sample sentence for testing."
print("Longest word in the sentence:", find_longest_word(user_sentence))
```

```
# 78. Sort a list using any one sorting technique.
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Example usage
unsorted_list = [64, 25, 12, 22, 11]
print("Unsorted list:", unsorted_list)
print("Sorted list using bubble sort:",
bubble_sort(unsorted_list))

# 79. Write a program to copy the contents of one text file to
another in Python.
def copy_file(source_file, destination_file):
    with open(source_file, 'r') as source:
        with open(destination_file, 'w') as destination:
            destination.write(source.read())
```

```
# Example usage
source_file = "source.txt"
destination_file = "destination.txt"
copy_file(source_file, destination_file)
print("Contents copied successfully.")


# 80. Program to count the occurrences of a specific character in a text file in Python.
def count_char_occurrences(filename, char):
    with open(filename, 'r') as file:
        content = file.read()
        return content.count(char)

# Example usage
text_file = "sample.txt"
character = "e"
print(f"Occurrences of '{character}' in the file:", count_char_occurrences(text_file, character))
```

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

```
def count_upper_lower(sentence):
    upper_count = sum(1 for char in sentence if char.isupper())
    lower_count = sum(1 for char in sentence if char.islower())
    return upper_count, lower_count


# Example usage
user_sentence = "Hello, World!"
upper, lower = count_upper_lower(user_sentence)
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_and_store(string1, string2, output_file):
    concatenated_string = string1 + string2
    with open(output_file, 'w') as file:
        file.write(concatenated_string)


# Example usage
```

```
input_string1 = "Hello, "

input_string2 = "World!"

output_file = "output.txt"

concatenate_and_store(input_string1, input_string2,
output_file)

print("Strings concatenated and stored in 'output.txt'.")


# 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):

    if (year % 4 == 0 and year % 100 != 0) or (year % 400 ==
0):

        return True

    else:

        return False


# Example usage

year = 2024

print(f"Is {year} a leap year?", is_leap_year(year))
```

84. Date difference calculator in python

```
def date_difference(date1, date2):
    return abs(date1 - date2).days


# Example usage
from datetime import datetime
date1 = datetime(2023, 5, 10)
date2 = datetime(2023, 5, 5)
print("Difference between the dates (in days):",
date_difference(date1, date2))
```

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

```
def km_to_miles(km):
    return km / 1.60934


# Example usage
kilometers = 100
print(f"{kilometers} kilometers is equal to
{km_to_miles(kilometers)} miles")
```

```
# 86. Python program to convert miles to kilometers
[Kilometers = Miles * 1.60934]

def miles_to_km(miles):
    return miles * 1.60934


# Example usage

miles = 62.137

print(f"{miles} miles is equal to {miles_to_km(miles)}
kilometers")


# 87. Calculate the area of a triangle in Python.

def triangle_area(base, height):
    return 0.5 *


 base * height


# Example usage

base = 6

height = 8

print("Area of the triangle:", triangle_area(base, height))
```

```
# 88. Python Program to Check if a Number is Odd or Even.
def check_odd_even(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

# Example usage
num = 7
print(f"{num} is {check_odd_even(num)}")

# 89. Python Program to Swap Two Variables.
def swap_variables(a, b):
    return b, a

# Example usage
var1 = 5
var2 = 10
print("Before swapping:", "var1 =", var1, "var2 =", var2)
var1, var2 = swap_variables(var1, var2)
print("After swapping:", "var1 =", var1, "var2 =", var2)
```

```
# 90. Python Program to Generate a Random Number
import random

def generate_random_number(start, end):
    return random.randint(start, end)

# Example usage
start_range = 1
end_range = 100
print("Random number between", start_range, "and", end_range, ":", generate_random_number(start_range, end_range))

# 91. Generate a random password which has the letters, digits, and special characters
import string

def generate_random_password(length):
    characters = string.ascii_letters + string.digits + string.punctuation
```

```
    return ''.join(random.choice(characters) for i in
range(length))


# Example usage

password_length = 10

print("Random password:",
generate_random_password(password_length))


# 92. Write a Python function to calculate the factorial of a
number (a non-negative integer).

def factorial(n):

    if n == 0:

        return 1

    else:

        return n * factorial(n - 1)


# Example usage

num = 5

print(f"Factorial of {num}:", factorial(num))


# 93. Python program to find the HCF or GCD of two
numbers
```

```
def calculate_hcf(num1, num2):
    while num2:
        num1, num2 = num2, num1 % num2
    return num1

# Example usage
num1 = 24
num2 = 36
print("GCD of", num1, "and", num2, ":", calculate_hcf(num1, num2))

# 94. Python program to make a simple calculator that can add, subtract, multiply, and divide.
def calculator(operation, num1, num2):
    if operation == 'add':
        return num1 + num2
    elif operation == 'subtract':
        return num1 - num2
    elif operation == 'multiply':
        return num1 * num2
    elif operation == 'divide':
```

```
        if num2 == 0:
            return "Cannot divide by zero"
        else:
            return num1 / num2
    else:
        return "Invalid operation"

# Example usage
operation = 'divide'
num1 = 10
num2 = 5
print("Result of", operation, "operation:",
calculator(operation, num1, num2))

# 95. Write a Python program to find the factors of a number.
def find_factors(number):
    factors = []
    for i in range(1, number + 1):
        if number % i == 0:
            factors.append(i)
    return factors
```

```
# Example usage
num = 24
print("Factors of", num, ":", find_factors(num))


# 96. Write a Python function to print the Fibonacci series up to n terms.
def fibonacci_series(n):
    fibonacci = [0, 1]
    for i in range(2, n):
        fibonacci.append(fibonacci[i-1] + fibonacci[i-2])
    return fibonacci


# Example usage
terms = 10
print("Fibonacci series up to", terms, "terms:", fibonacci_series(terms))


# 97. Python program to find the sum of digits in a number
def sum_of_digits(number):
    return sum(int(digit) for digit in str(number))
```

```
# Example usage
num = 12345
print("Sum of digits in", num, ":", sum_of_digits(num))

# 98. Python program to find the reverse of a number
def reverse_number(number):
    return int(str(number)[::-1])

# Example usage
num = 12345
print("Reverse of", num, ":", reverse_number(num))

# 99. Python program to find whether a number is palindrome or not
def is_palindrome(number):
    return str(number) == str(number)[::-1]

# Example usage
num = 12321
print(num, "is a palindrome?" , is_palindrome(num))
```

100. Python program to find the sum of elements in a list recursively.

```
def sum_recursive(lst):
    if len(lst) == 1:
        return lst[0]
    else:
        return lst[0] + sum_recursive(lst[1:])

# Example usage
numbers = [1, 2, 3, 4, 5]
print("Sum of elements in the list recursively:", sum_recursive(numbers))
```

101. Python program to find the factorial of a number using recursion.

```
def factorial_recursive(n):
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)
```

```
# Example usage
num = 5
print("Factorial of", num, ":", factorial_recursive(num))


# 102. Python program to find the Fibonacci series using recursion
def fibonacci_recursive(n):
    if n <= 1:
        return n
    else:
        return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)

# Example usage
terms = 10
print("Fibonacci series up to", terms, "terms:")
for i in range(terms):
    print(fibonacci_recursive(i), end=" ")
```

103. Python program to count the number of occurrences of a character in a string recursively.

```
def count_occurrences_recursive(string, char):
    if not string:
        return 0
    elif string[0] == char:
        return 1 + count_occurrences_recursive(string[1:], char)
    else:
        return count_occurrences_recursive(string[1:], char)

# Example usage
user_string = "Hello, World!"
character = "l"
print(f"Number of occurrences of '{character}' in the string:", count_occurrences_recursive(user_string, character))
```

104. Python program to find the length of a string using recursion.

```
def string_length_recursive(string):
    if not string:
        return 0
```

```
    else:
        return 1 + string_length_recursive(string[1:])

# Example usage
user_string = "Hello, World!"
print("Length of the string:",
string_length_recursive(user_string))

# 105. Python program to find the maximum element in a list
using recursion.
def max_in_list_recursive(lst):
    if len(lst) == 1:
        return lst[0]
    else:
        return max(lst[0], max_in_list_recursive(lst[1:]))

# Example usage
numbers = [5, 8, 1, 3, 9, 2]
print("Maximum element in the list

:", max_in_list_recursive(numbers))
```

106. Python program to find the power of a number using recursion.

```
def power_recursive(base, exponent):
    if exponent == 0:
        return 1
    elif exponent == 1:
        return base
    else:
        return base * power_recursive(base, exponent - 1)

# Example usage
base = 2
exponent = 5
print(f"{base} raised to the power of {exponent}:", power_recursive(base, exponent))
```

107. Python program to perform a binary search recursively.

```
def binary_search_recursive(arr, low, high, target):
    if high >= low:
        mid = (high + low) // 2
```

```
        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            return binary_search_recursive(arr, low, mid - 1,
target)
        else:
            return binary_search_recursive(arr, mid + 1, high,
target)
    else:
        return -1

# Example usage
sorted_array = [2, 3, 4, 10, 40]
target_element = 10
index = binary_search_recursive(sorted_array, 0,
len(sorted_array)-1, target_element)
if index != -1:
    print(f"Element {target_element} is present at index
{index}")
else:
    print(f"Element {target_element} is not present in the
array")
```

```
# 108. Python program to perform a linear search recursively.
def linear_search_recursive(arr, index, target):
    if index == len(arr):
        return -1
    elif arr[index] == target:
        return index
    else:
        return linear_search_recursive(arr, index + 1, target)

# Example usage
array = [4, 2, 7, 1, 9, 5]
target_value = 7
index = linear_search_recursive(array, 0, target_value)
if index != -1:
    print(f"Element {target_value} is present at index {index}")
else:
    print(f"Element {target_value} is not present in the array")

# 109. Python program to perform insertion sort recursively.
```

```
def insertion_sort_recursive(arr):
    if len(arr) <= 1:
        return arr
    else:
        sorted_arr = insertion_sort_recursive(arr[1:])
        key = arr[0]
        i = len(sorted_arr) - 1
        while i >= 0 and sorted_arr[i] > key:
            sorted_arr[i + 1] = sorted_arr[i]
            i -= 1
        sorted_arr[i + 1] = key
        return sorted_arr

# Example usage
unsorted_array = [12, 11, 13, 5, 6]
print("Original array:", unsorted_array)
sorted_array = insertion_sort_recursive(unsorted_array)
print("Sorted array:", sorted_array)

# 110. Python program to perform selection sort recursively.
def selection_sort_recursive(arr):
```

```
    if len(arr) <= 1:
        return arr
    else:
        min_index = arr.index(min(arr))
        arr[0], arr[min_index] = arr[min_index], arr[0]
        return [arr[0]] + selection_sort_recursive(arr[1:])

# Example usage
unsorted_array = [64, 25, 12, 22, 11]
print("Original array:", unsorted_array)
sorted_array = selection_sort_recursive(unsorted_array)
print("Sorted array:", sorted_array)

# 111. Python program to perform bubble sort recursively.
def bubble_sort_recursive(arr):
    n = len(arr)
    if n <= 1:
        return arr
    else:
        for i in range(n-1):
            if arr[i] > arr[i+1]:
```

```
                arr[i], arr[i+1] = arr[i+1], arr[i]
        return bubble_sort_recursive(arr[:-1]) + [arr[-1]]

# Example usage
unsorted_array = [64, 25, 12, 22, 11]
print("Original array:", unsorted_array)
sorted_array = bubble_sort_recursive(unsorted_array)
print("Sorted array:", sorted_array)

# 112. Write a Python program to perform a linear search.
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

# Example usage
array = [4, 2, 7, 1, 9, 5]
target_value = 7
index = linear_search(array, target_value)
if index != -1:
```

```
    print(f"Element {target_value} is present at index
{index}")
else:
    print(f"Element {target_value} is not present in the array")


# 113. Python program to perform a binary search.
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1


# Example usage
sorted_array = [2, 3, 4, 10, 40]
```

```
target_element = 10
index = binary_search(sorted_array, target_element)
if index != -1:
    print(f"Element {target_element} is present at index {index}")
else:
    print(f"Element {target_element} is not present in the array")


# 114. Python program to perform an insertion sort.
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# Example usage
unsorted_array = [12, 11, 13, 5, 6]
```

```
print("Original array:", unsorted_array)
insertion_sort(unsorted_array)
print("Sorted array:", unsorted_array)

# 115. Python program to perform a selection sort.
def selection_sort(arr):
    for i in range(len(arr)):
        min_index = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]

# Example usage
unsorted_array = [64, 25, 12, 22, 11]
print("Original array:", unsorted_array)
selection_sort(unsorted_array)
print("Sorted array:", unsorted_array)

# 116. Python program to perform a bubble sort.
def bubble_sort(arr):
```

```
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Example usage
unsorted_array = [64, 25, 12, 22, 11]
print("Original array:", unsorted_array)
bubble_sort


(unsorted_array)
print("Sorted array:", unsorted_array)


# 117. Python program to implement the insertion sort algorithm for sorting elements in a list.
def insertion_sort(lst):
    for i in range(1, len(lst)):
        key = lst[i]
        j = i - 1
        while j >= 0 and lst[j] > key:
```

```
            lst[j + 1] = lst[j]
            j -= 1
        lst[j + 1] = key

# Example usage
my_list = [12, 11, 13, 5, 6]
print("Original list:", my_list)
insertion_sort(my_list)
print("Sorted list:", my_list)

# 118. Python program to implement the selection sort algorithm for sorting elements in a list.
def selection_sort(lst):
    for i in range(len(lst)):
        min_idx = i
        for j in range(i+1, len(lst)):
            if lst[j] < lst[min_idx]:
                min_idx = j
        lst[i], lst[min_idx] = lst[min_idx], lst[i]

# Example usage
```

```
my_list = [64, 25, 12, 22, 11]
print("Original list:", my_list)
selection_sort(my_list)
print("Sorted list:", my_list)
```

119. Python program to implement the bubble sort algorithm for sorting elements in a list.

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

# Example usage
my_list = [64, 25, 12, 22, 11]
print("Original list:", my_list)
bubble_sort(my_list)
print("Sorted list:", my_list)
```

120. Python program to implement the quick sort algorithm for sorting elements in a list.

```
def quick_sort(lst):
    if len(lst) <= 1:
        return lst
    pivot = lst[len(lst) // 2]
    left = [x for x in lst if x < pivot]
    middle = [x for x in lst if x == pivot]
    right = [x for x in lst if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# Example usage
my_list = [3, 6, 8, 10, 1, 2, 1]
print("Original list:", my_list)
sorted_list = quick_sort(my_list)
print("Sorted list:", sorted_list)
```

121. Python program to implement the merge sort algorithm for sorting elements in a list.

```
def merge_sort(lst):
    if len(lst) <= 1:
```

```
        return lst
    mid = len(lst) // 2
    left_half = merge_sort(lst[:mid])
    right_half = merge_sort(lst[mid:])
    return merge(left_half, right_half)

def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0
    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx] < right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1
    result.extend(left[left_idx:])
    result.extend(right[right_idx:])
    return result

# Example usage
```

```
my_list = [3, 6, 8, 10, 1, 2, 1]
print("Original list:", my_list)
sorted_list = merge_sort(my_list)
print("Sorted list:", sorted_list)
```

122. Python program to implement the heap sort algorithm for sorting elements in a list.

```
def heap_sort(lst):
    n = len(lst)
    for i in range(n // 2 - 1, -1, -1):
        heapify(lst, n, i)
    for i in range(n-1, 0, -1):
        lst[i], lst[0] = lst[0], lst[i]
        heapify(lst, i, 0)

def heapify(lst, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2
    if left < n and lst[left] > lst[largest]:
        largest = left
```

```
    if right < n and lst[right] > lst[largest]:
        largest = right
    if largest != i:
        lst[i], lst[largest] = lst[largest], lst[i]
        heapify(lst, n, largest)

# Example usage
my_list = [12, 11, 13, 5, 6, 7]
print("Original list:", my_list)
heap_sort(my_list)
print("Sorted list:", my_list)

# 123. Python program to implement the shell sort algorithm for sorting elements in a list.
def shell_sort(lst):
    n = len(lst)
    gap = n // 2
    while gap > 0:
        for i in range(gap, n):
            temp = lst[i]
            j = i
```

```
            while j >= gap and lst[j - gap] > temp:
                lst[j] = lst[j - gap]
                j -= gap
            lst[j] = temp
        gap //= 2

# Example usage
my_list = [12, 34, 54, 2, 3]
print("Original list:", my_list)
shell_sort(my_list)
print("Sorted list:", my_list)
```

124. Python program to find the square root of a number without using any built-in functions.

```
def sqrt_without_builtin(number):
    if number < 0:
        return "Invalid input"
    guess = number / 2
    while abs(guess * guess - number) > 0.0001:
        guess = (guess + number / guess) / 2
    return guess
```

```
# Example usage
num = 16
print("Square root of", num, ":", sqrt_without_builtin(num))


# 125. Python program to find the cube root of a number without using any built-in functions.
def cube_root_without_builtin(number):
    if number < 0:
        return -(-number) ** (1. / 3)
    else:
        return number ** (1. / 3)

# Example usage
num = 27
print("Cube root of", num, ":",
cube_root_without_builtin(num))


# 126. Python program to perform matrix multiplication.
def matrix_multiplication(matrix1, matrix2):
```

```
    result = [[0 for _ in range(len(matrix2[0]))] for _ in
range(len(matrix1))]
    for i in range(len(matrix1)):
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):
                result[i][j] += matrix1[i][k] * matrix2[k][j]
    return result

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

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

result = matrix

_multiplication(matrix1, matrix2)
print("Result of matrix multiplication:")
```

```
for row in result:
    print(row)


# 127. Python program to find the transpose of a matrix.
def transpose_matrix(matrix):
    return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]


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

transposed_matrix = transpose_matrix(matrix)
print("Transposed matrix:")
for row in transposed_matrix:
    print(row)


# 128. Python program to add two matrices.
def add_matrices(matrix1, matrix2):
```

```
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        return "Matrices must have the same dimensions for addition."
    else:
        return [[matrix1[i][j] + matrix2[i][j] for j in range(len(matrix1[0]))] for i in range(len(matrix1))]

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

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

result = add_matrices(matrix1, matrix2)
print("Result of matrix addition:")
for row in result:
    print(row)
```

```
# 129. Python program to subtract two matrices.

def subtract_matrices(matrix1, matrix2):
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        return "Matrices must have the same dimensions for subtraction."
    else:
        return [[matrix1[i][j] - matrix2[i][j] for j in range(len(matrix1[0]))] for i in range(len(matrix1))]


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

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

result = subtract_matrices(matrix1, matrix2)
```

```
print("Result of matrix subtraction:")
for row in result:
    print(row)


# 130. Python program to find the length of the longest consecutive sequence of elements in an unsorted array.
def longest_consecutive_sequence(arr):
    if not arr:
        return 0
    num_set = set(arr)
    max_length = 0
    for num in num_set:
        if num - 1 not in num_set:
            current_num = num
            current_length = 1
            while current_num + 1 in num_set:
                current_num += 1
                current_length += 1
            max_length = max(max_length, current_length)
    return max_length
```

```
# Example usage
sequence = [100, 4, 200, 1, 3, 2]
print("Length of the longest consecutive sequence:", longest_consecutive_sequence(sequence))


# 131. Python program to find the first missing positive integer in an unsorted array.
def first_missing_positive(nums):
    if not nums:
        return 1
    nums_set = set(nums)
    for i in range(1, len(nums) + 1):
        if i not in nums_set:
            return i
    return len(nums) + 1


# Example usage
array = [3, 4, -1, 1]
print("First missing positive integer:", first_missing_positive(array))
```

132. Python program to implement a stack using a linked list.

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class Stack:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head is None

    def push(self, value):
        new_node = Node(value)
        new_node.next = self.head
        self.head = new_node

    def pop(self):
        if self.is_empty():
```

```
            return "Stack is empty"
        else:
            popped_value = self.head.value
            self.head = self.head.next
            return popped_value

    def peek(self):
        if self.is_empty():
            return "Stack is empty"
        else:
            return self.head.value

# Example usage
stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print("Top element of the stack:", stack.peek())
print("Popped element:", stack.pop())
print("Popped element:", stack.pop())
print("Is stack empty?", stack.is_empty())
```

133. Python program to implement a queue using two stacks.

```
class Queue:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, value):
        self.stack1.append(value)

    def dequeue(self):
        if not self.stack2:
            if not self.stack1:
                return "Queue is empty"
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop()

# Example usage
queue = Queue()
```

```
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print("Dequeued element:", queue.dequeue())
print("Dequeued element:", queue.dequeue())

# 134. Python program to reverse a stack using recursion.
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
```

```
        return self.items[-1]

def insert_at_bottom(stack, item):
    if stack.is_empty():
        stack.push(item)
    else:
        temp = stack.pop()
        insert_at_bottom(stack, item)
        stack.push(temp)

def reverse_stack(stack):
    if not stack.is_empty():
        temp = stack.pop()
        reverse_stack(stack)
        insert_at_bottom(stack, temp)

# Example usage
stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
```

```
print("Original stack:", stack.items)
reverse_stack(stack)
print("Reversed stack:", stack.items)

# 135. Python program to implement a deque (double-ended queue).
class Deque:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def add_front(self, item):
        self.items.append(item)

    def add_rear(self, item):
        self.items.insert(0, item)

    def remove_front(self):
        if self.is_empty():
```

```
            return "Deque is empty"
        return self.items.pop()

    def remove_rear(self):
        if self.is_empty():
            return "Deque is empty"
        return self.items.pop(0)

    def peek_front(self):
        if self.is_empty():
            return "Deque is empty"
        return self.items[-1]

    def peek_rear(self):
        if self.is_empty():
            return "Deque is empty"
        return self.items[0]

    def size(self):
        return len(self.items)
```

```
#


 Example usage
deque = Deque()
deque.add_front(1)
deque.add_front(2)
deque.add_rear(3)
deque.add_rear(4)
print("Deque size:", deque.size())
print("Front of deque:", deque.peek_front())
print("Rear of deque:", deque.peek_rear())
print("Removing front:", deque.remove_front())
print("Removing rear:", deque.remove_rear())


# 136. Python program to implement a circular queue.
class CircularQueue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = self.rear = -1
```

```
def enqueue(self, item):
    if (self.rear + 1) % self.size == self.front:
        return "Queue is full"
    elif self.front == -1:
        self.front = 0
        self.rear = 0
        self.queue[self.rear] = item
    else:
        self.rear = (self.rear + 1) % self.size
        self.queue[self.rear] = item

def dequeue(self):
    if self.front == -1:
        return "Queue is empty"
    elif self.front == self.rear:
        temp = self.queue[self.front]
        self.front = -1
        self.rear = -1
        return temp
    else:
        temp = self.queue[self.front]
```

```
            self.front = (self.front + 1) % self.size
            return temp

    def display(self):
        if self.front == -1:
            return "Queue is empty"
        elif self.rear >= self.front:
            return self.queue[self.front:self.rear + 1]
        else:
            return self.queue[self.front:self.size] +
self.queue[0:self.rear + 1]

# Example usage
cq = CircularQueue(5)
cq.enqueue(1)
cq.enqueue(2)
cq.enqueue(3)
cq.enqueue(4)
cq.enqueue(5)
print("Initial queue:", cq.display())
cq.dequeue()
```

```
cq.dequeue()
print("After removing two elements:", cq.display())
cq.enqueue(6)
print("After adding one element:", cq.display())

# 137. Python program to perform a breadth-first search (BFS) traversal of a graph.
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
        self.graph[u].append(v)

    def bfs(self, start):
        visited = [False] * (max(self.graph) + 1)
        queue = []
        queue.append(start)
        visited[start] = True
```

```
        while queue:
            start = queue.pop(0)
            print(start, end=" ")

            for i in self.graph[start]:
                if not visited[i]:
                    queue.append(i)
                    visited[i] = True

# Example usage
g = Graph()
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 2)
g.add_edge(2, 0)
g.add_edge(2, 3)
g.add_edge(3, 3)

print("Breadth First Traversal (starting from vertex 2):")
g.bfs(2)
```

138. Python program to perform a depth-first search (DFS) traversal of a graph.

```
from collections import defaultdict


class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
        self.graph[u].append(v)

    def dfs_util(self, v, visited):
        visited.add(v)
        print(v, end=" ")

        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.dfs_util(neighbour, visited)

    def dfs(self, start):
```

```
        visited = set()
        self.dfs_util(start, visited)

# Example usage
g = Graph()
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 2)
g.add_edge(2, 0)
g.add_edge(2, 3)
g.add_edge(3, 3)

print("Depth First Traversal (starting from vertex 2):")
g.dfs(2)

# 139. Python program to implement Dijkstra's algorithm for finding the shortest path in a graph.
from collections import defaultdict

class Graph:
    def __init__(self):
```

```
        self.graph = defaultdict(dict)


    def add_edge(self, u, v, w):
        self.graph[u][v] = w


    def dijkstra(self, start):
        nodes = list(self.graph.keys())
        visited = set()
        distances = {node: float('inf') for node in nodes}
        distances[start] = 0


        while len(visited) < len(nodes):
            current_node = min({node: distances[node] for node
in nodes if node not in visited}, key=distances.get)
            visited.add(current_node)
            for neighbour, weight in
self.graph[current_node].items():
                if distances[current_node] + weight <
distances[neighbour]:
                    distances[neighbour] = distances[current_node] +
weight
```

```
        return distances

# Example usage
g = Graph()
g.add_edge('A', 'B', 4)
g.add_edge('A', 'C', 2)
g.add_edge('B', 'C', 5)
g.add_edge('B', 'D', 10)
g.add_edge('C', 'D', 3)

start_node = 'A'
print("Shortest distances from node", start_node, ":")
print(g.dijkstra(start_node))

# 140. Python program to find all paths between two nodes in a graph.
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
```

```
        self.graph[u].append(v)

    def find_all_paths(self, start, end, path=[]):
        path = path + [start]
        if start == end:
            return [path]
        paths = []
        for node in self.graph[start]:
            if node not in path:
                new_paths = self.find_all_paths(node, end, path)
                for new_path in new_paths:
                    paths.append(new_path)
        return paths

# Example usage
g = Graph()
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 2)
g.add_edge(2, 0)
g.add_edge(2, 3)
```

```
g.add_edge(3, 3)

start_node = 2
end_node = 3
print("All paths from", start_node, "to", end_node, ":")
print(g.find_all_paths(start_node, end_node))
```