## 1) Implement factorial using recursion

```
def factorial (n): if n == 0:
   return 1
   else:
   return n factorial(n - 1)
   n=5
   result=factorial (n)
   print (f" {n}!= {result)}")
```

## 2) Program using doubly linked list

**class Node:**

```
def__init__(self, data):
self.data = data
self.prev=None
self.next=None
class DoublyLinkedList:
def __init_(self):
self.head= None
self.tail= None
def append(self, data):
new_node=Node(data)
if self.head is None:
self.head new_node
self.tail = new_node
else:
new_node.prev= self.tail self.tail
self.tail.next = new_node
self.tail= new_node

def prepend(self, data):
new_node = Node(data)
ifself.head is None:
self.head = new_node
self.tail = new_node
else:
new_node.next = self.head
self.head.prev = new_node
self.head= new_node
def print_list(self):
current = self.head
while current:
```

```
print(current.data)
current = current.next
# Example usage

my_list = DoublyLinkedList()
my_list.append(1)
my list.append(2)
my list.append(3)
my_list.prepend(0)
my_list.print_list()
```

## 3) Implement multiple inheritance using interface

```
class A:
def method_a(self):
print("This is method A")
class B:
def method_b(self):
print("This is method B")
class C(A, B):
pass
class D(B, A):
pass
```

## 4) Print all pronic numbers between 1 and 100

```
def is pronic(n):
for i in range(1, n):
if i(i+1)=n:
return True
return False
for i in range(1, 101):
 if is pronic(i):
print(i)
```

## 5) Implement method overloading & overiding in python

```
class MyClass:
   def my_method(self, x, y=0):
      if y == 0:
      print("The value of x is", x)
      else:
```

```
      print("The values of x and y are", x, "and", y)

      obj = MyClass()

     obj.my_method(10)

    obj.my_method(10, 20)
```

6) Program to find duplicate values for ArrayList

```
   my_list = [1, 2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9]
  seen = set()
 duplicates = []
 for element in my_list:
 if element in seen:
   duplicates.append(element)
else:
   seen.add(element)

print("The duplicates in the list are:", duplicates)
```

7) Python program to print the elements of an array in reverse order

```
arr = [1, 2, 3, 4, 5]
arr.reverse()
print("Reversed array:", arr)
```

8) Python program to determine whether the given number is a Harshad Number

```
num = int(input("Enter a number: "))
sum_of_digits = sum([int(d) for d in str(num)])
if num % sum_of_digits == 0:
   print(num, "is a Harshad number")
else:
   print(num, "is not a Harshad number")
```

9) Implement a program to merge two Arrays

```
arr1 = [1, 2, 3]
arr2 = [4, 5, 6]
merged_arr = arr1 + arr2
```

```
print("Merged array using + operator:", merged_arr)
arr3 = [7, 8, 9]
arr4 = [10, 11, 12]
arr3.extend(arr4)
print("Merged array using extend() method:", arr3)
```

10) Program to find duplicate values for ArrayList

```
arr = [1, 2, 3, 4, 3, 2, 1, 5, 6, 7, 5]
unique_elems = set()
duplicates = []
for elem in arr:
   if elem in unique_elems:
      duplicates.append(elem)
   else:
      unique_elems.add(elem)
print("Duplicate elements:", duplicates)
```

11) Implement a program to sort a map by value / Key

```
# Create a map with some key-value pairs

my_map = {"apple": 5, "banana": 2, "orange": 4, "kiwi": 1}

sorted_by_value = dict(sorted(my_map.items(), key=lambda item: item[1]))

print("Sorted by value:", sorted_by_value)

sorted_by_key = dict(sorted(my_map.items(), key=lambda item: item[0]))

print("Sorted by key:", sorted_by_key)
```

12) Write a python Program for Fibonacci series.

```
def fibonacci(n):

fib = [0, 1]    # Start with the first two terms

for i in range(2, n):

fib.append(fib[i-1]  +  fib[i-2])    # Add the previous two terms to get the next term

return fib

print(fibonacci(10))    # prints [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

13) Python program to print the elements of an array in reverse order

```
arr = [1, 2, 3, 4, 5]
for i in range(len(arr)-1, -1, -1):
   print(arr[i])
arr.reverse()
print(arr)
```

14) Python program for Fibonacci

```
  def fibonacci(n):

 fib = [0, 1]     # Start with the first two terms

 for i in range(2, n):

fib.append(fib[i-1]  +  fib[i-2])     # Add the previous two terms to get the next term

return fib

print(fibonacci(10))     # prints [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

15) Constructor Overloading

```
 class Person:
   def __init__(self, name=None, age=None):
     if name is not None and age is not None:
       self.name = name
       self.age = age
     elif name is not None:
       self.name = name
       self.age = 0
     else:
       self.name = "Unknown"
       self.age = 0

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

# Example usage:
p1 = Person("John", 30)
p2 = Person("Jane")
p3 = Person()

p1.display()  # Name: John, Age: 30
p2.display()  # Name: Jane, Age: 0
p3.display()  # Name: Unknown, Age: 0
```

16)
```
 arr = [1, 2, 3, 4, 5]

# Using reverse() method
arr.reverse()

# Using slicing
reverse_arr = arr[::-1]

# Using a loop
for i in range(len(arr)-1, -1, -1):
    print(arr[i])
```

17)
```
      try:
    # code that may raise an exception
    x = 10 / 0
finally:
    # code that will always be executed
    print("This is the finally block")
```

18)
```
    num = int(input("Enter a number: "))
sum_of_digits = sum([int(d) for d in str(num)])
if num % sum_of_digits == 0:
    print(num, "is a Harshad number")
else:
    print(num, "is not a Harshad number")
```

19)
```
    s1 = "Hello"
s2 = "Hello"
s3 = "World"

if s1 == s2:
    print("s1 and s2 are equal")
else:
    print("s1 and s2 are not equal")

if s1 is s2:
    print("s1 and s2 are the same object")
else:
    print("s1 and s2 are not the same object")

if s1 == s3:
    print("s1 and s3 are equal")
else:
    print("s1 and s3 are not equal")
```

20) Python program to print the elements of an array in reverse order

```
arr = [1, 2, 3, 4, 5]
arr.reverse()
print("Reversed array:", arr)
```

21)

```
    def remove_duplicates(arr):
    n = len(arr)
    if n == 0 or n == 1:
        return n

    i = 0
    for j in range(1, n):
        if arr[j] != arr[i]:
            i += 1
            arr[i] = arr[j]

    return i+1

# example usage
arr = [1, 2, 2, 3, 3, 3, 4, 5, 5]
new_length = remove_duplicates(arr)
print(arr[:new_length])
```

22) Python program to print the elements of an array in reverse order

```
arr = [1, 2, 3, 4, 5]
arr.reverse()
print("Reversed array:", arr)
```

23)

```
    # initial values of variables
    a = 5
```

```
b = 7

# swapping without a temporary variable
a, b = b, a

# printing the swapped values
print("a =", a)
print("b =", b)
```

24)

```
 class Node:
  def __init__(self, val):
    self.val = val
    self.left = None
    self.middle = None
    self.right = None

class DoublyLinkedListNode:
  def __init__(self, val):
    self.val = val
    self.prev = None
    self.next = None

def create_doubly_linked_list(root):
  if not root:
    return None
  head = create_doubly_linked_list(root.left)
  middle_head = create_doubly_linked_list(root.middle)
  tail = create_doubly_linked_list(root.right)
  new_node = DoublyLinkedListNode(root.val)
  if not head:
    head = new_node
  else:
    tail_node = head
    while tail_node.next:
      tail_node = tail_node.next
    new_node.prev = tail_node
    tail_node.next = new_node
  if middle_head:
    tail_node = head
    while tail_node.next:
      tail_node = tail_node.next
    tail_node.next = middle_head
```

```
            middle_head.prev = tail_node
        if tail:
            tail_node = head
            while tail_node.next:
                tail_node = tail_node.next
            tail_node.next = tail
            tail.prev = tail_node
        return head
```

25)

```
def max_repeated_char_count(string):
char_freq = {}  # create an empty dictionary to store the frequency of each character
    for char in string:
    if char in char_freq:
        char_freq[char] += 1
    else:
        char_freq[char] = 1

max_freq = 0
max_char = ''
# iterate over the dictionary to find the character with maximum frequency
for char, freq in char_freq.items():
    if freq > max_freq:
        max_freq = freq
        max_char = char

return max_freq
```

26)

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]  # choose the first element as the pivot
        left = []
        right = []
        middle = []

        for num in arr:
            if num < pivot:
                left.append(num)
            elif num > pivot:
                right.append(num)
        else:
            middle.append(num)

    return quick_sort(left) + middle + quick_sort(right)
```

27)

```
def find_duplicates(s):
count_dict = {}
duplicates = []

for char in s:
    if char in count_dict:
        count_dict[char] += 1
    else:
        count_dict[char] = 1

for char, count in count_dict.items():
    if count > 1:
        duplicates.append(char)

return duplicates
```

28)

```
 num = int(input("Enter a number: "))
sum_of_digits = sum([int(d) for d in str(num)])
if num % sum_of_digits == 0:
    print(num, "is a Harshad number")
else:
    print(num, "is not a Harshad number")
```

29)

```
 def sum_of_list(lst):
    total = 0
    for num in lst:
        total += num
    return total

my_list = [1, 2, 3, 4, 5]
print(sum_of_list(my_list))  o/p sum =15
```

30)

```
    my_dict = {}

my_dict['apple'] = 2
my_dict['banana'] = 3
my_dict['orange'] = 1

print(my_dict['apple'])  # output: 2
print(my_dict['banana'])  # output: 3

for key in my_dict:
    print(key)  # output: apple, banana, orange

for key, value in my_dict.items():
    print(key, value)  # output: apple 2, banana 3, orange 1

if 'apple' in my_dict:
    print('Yes')  # output: Yes

del my_dict['orange']
print(my_dict)  # output: {'apple': 2, 'banana': 3}
```

31)

```
     num = int(input("Enter a number: "))
sum_of_digits = sum([int(d) for d in str(num)])
if num % sum_of_digits == 0:
    print(num, "is a Harshad number")
else:
    print(num, "is not a Harshad number")
```

32)

```
     class Node:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.middle = None
        self.right = None

class DoublyLinkedListNode:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None

def create_doubly_linked_list(root):
    if not root:
        return None
    head = create_doubly_linked_list(root.left)
    middle_head = create_doubly_linked_list(root.middle)
    tail = create_doubly_linked_list(root.right)
    new_node = DoublyLinkedListNode(root.val)
    if not head:
        head = new_node
    else:
        tail_node = head
        while tail_node.next:
            tail_node = tail_node.next
        new_node.prev = tail_node
        tail_node.next = new_node
    if middle_head:
        tail_node = head
        while tail_node.next:
            tail_node = tail_node.next
        tail_node.next = middle_head
        middle_head.prev = tail_node
    if tail:
        tail_node = head
        while tail_node.next:
            tail_node = tail_node.next
        tail_node.next = tail
        tail.prev = tail_node
    return head
```

33)

```
    array1 = [1, 2, 3, 4, 5]
array2 = [3, 4, 5, 6, 7]

common_elements = []

for element in array1:
    if element in array2 and element not in common_elements:
        common_elements.append(element)

print(common_elements)      output: [3, 4, 5]
```

34)

```
   def is_palindrome(s):
   # Convert to string if input is a number
   s = str(s)
   # Remove spaces and convert to lowercase
   s = s.replace(" ", "").lower()
   # Check if string is equal to its reverse
   return s == s[::-1]

# Example usage
print(is_palindrome("racecar"))  # Output: True
print(is_palindrome(12321))  # Output: True
print(is_palindrome("Hello World"))  # Output: False
```

35)

```
   from abc import ABC, abstractmethod

class Interface1(ABC):
   @abstractmethod
   def method1(self):
      pass

class Interface2(ABC):
   @abstractmethod
   def method2(self):
      pass

class MyClass(Interface1, Interface2):
   def method1(self):
      print("Method 1 called")

   def method2(self):
      print("Method 2 called")
```

36)

```
num = int(input("Enter a number: "))
sum_of_digits = sum([int(d) for d in str(num)])
if num % sum_of_digits == 0:
print(num, "is a Harshad number")
else:
 print(num, "is not a Harshad number")
```

37)

```
  class Employee:
   def __init__(self, name, age, salary):
       self.__name = name
       self.__age = age
       self.__salary = salary

   def display_details(self):
       print("Name:", self.__name)
       print("Age:", self.__age)
       print("Salary:", self.__salary)

   def get_name(self):
       return self.__name

   def set_name(self, name):
       self.__name = name

   def get_age(self):
       return self.__age

   def set_age(self, age):
       self.__age = age

   def get_salary(self):
       return self.__salary

   def set_salary(self, salary):
       self.__salary = salary


emp = Employee("John Doe", 30, 50000)

emp.display_details()


emp.set_name("Jane Doe")


print("Updated Name:", emp.get_name())
```

38)

```
 def is pronic(n):
for i in range(1, n):
if i(i+1)=n:
return True
return False
for i in range(1, 101):
 if is pronic(i):
print(i)
```

39)

```
s = "hello"
c = s[0]
print(c)    # Output: 'h'

c = 'a'
s = c + ''
print(s)    # Output: 'a'
```

40)

```
  from collections import OrderedDict

# Initialize an ordered dictionary
od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3

# Iterate over the values
for value in od.values():
    print(value)
```

41)

```
 import abc

class Shape(metaclass=abc.ABCMeta):
   @abc.abstractmethod
   def area(self):
      pass

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

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

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

   def area(self):
      return 3.14 * self.radius * self.radius

r = Rectangle(5, 10)
c = Circle(7)

print("Area of rectangle:", r.area())
print("Area of circle:", c.area())
```

42)

```
   def is pronic(n):
for i in range(1, n):
if i(i+1)=n:
return True
return False
for i in range(1, 101):
 if is pronic(i):
print(i)
```

43)

```
   try:
   x = int(input("Enter a number: "))
   y = int(input("Enter another number: "))
   result = x / y
   print("Result:", result)
except ValueError:
   print("Invalid input, please enter a number")
except ZeroDivisionError:
   print("Cannot divide by zero")
except:
   print("An unknown error occurred")
```

44)

```
      class Node:
   def __init__(self, val):
      self.val = val
      self.left = None
      self.middle = None
      self.right = None

class DoublyLinkedListNode:
   def __init__(self, val):
      self.val = val
      self.prev = None
      self.next = None

def create_doubly_linked_list(root):
   if not root:
      return None
   head = create_doubly_linked_list(root.left)
   middle_head = create_doubly_linked_list(root.middle)
   tail = create_doubly_linked_list(root.right)
   new_node = DoublyLinkedListNode(root.val)
   if not head:
      head = new_node
   else:
      tail_node = head
      while tail_node.next:
         tail_node = tail_node.next
      new_node.prev = tail_node
```

```
        tail_node.next = new_node
    if middle_head:
        tail_node = head
        while tail_node.next:
            tail_node = tail_node.next
        tail_node.next = middle_head
        middle_head.prev = tail_node
    if tail:
        tail_node = head
        while tail_node.next:
            tail_node = tail_node.next
        tail_node.next = tail
        tail.prev = tail_node
    return head
```

45)

```
   my_list = ['apple', 'banana', 'orange']
my_string = ', '.join(my_list)
print(my_string)
```

46)

```
  num = int(input("Enter a number: "))
sum_of_digits = sum([int(d) for d in str(num)])
if num % sum_of_digits == 0:
print(num, "is a Harshad number")
else:
 print(num, "is not a Harshad number")
```

47)

```
   my_set = {1, 2, 3, 4, 5}
stream = []

for i in my_set:
    stream.append(i)

print(stream)
```

48)

```
     class Node:
  def __init__(self, val):
    self.val = val
    self.left = None
    self.middle = None
    self.right = None

class DoublyLinkedListNode:
  def __init__(self, val):
    self.val = val
    self.prev = None
    self.next = None

def create_doubly_linked_list(root):
  if not root:
    return None
  head = create_doubly_linked_list(root.left)
  middle_head = create_doubly_linked_list(root.middle)
  tail = create_doubly_linked_list(root.right)
  new_node = DoublyLinkedListNode(root.val)
  if not head:
    head = new_node
  else:
    tail_node = head
    while tail_node.next:
      tail_node = tail_node.next
    new_node.prev = tail_node
    tail_node.next = new_node
  if middle_head:
    tail_node = head
    while tail_node.next:
      tail_node = tail_node.next
    tail_node.next = middle_head
    middle_head.prev = tail_node
  if tail:
    tail_node = head
    while tail_node.next:
      tail_node = tail_node.next
    tail_node.next = tail
    tail.prev = tail_node
  return head
```

49)
```
 def is_palindrome(num):
"""
A recursive function to check if a number is palindrome or not.
"""
if num // 10 == 0:
    return True
else:
          first_digit = num % 10
    last_digit = num // (10 ** (len(str(num)) - 1))

    if first_digit != last_digit:
        return False
    else:

        return is_palindrome((num % (10 ** (len(str(num)) - 1))) // 10)

num = int(input("Enter a number to check if it's palindrome or not: "))
if is_palindrome(num):
    print(num, "is a palindrome.")
else:
    print(num, "is not a palindrome.")
```

50)

```
a = 10
b = 20

print("Before swapping a = ", a, " and b =", b)

a = a + b
b = a - b
a = a - b

print("After swapping a = ", a, " and b =", b)
```

51)
```
 def is_prime(n):
if n < 2:
    return False
for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
        return False
return True

def print_primes(start, end):
for n in range(start, end+1):
    if is_prime(n):
        print(n)

print_primes(10, 50)  # prints: 11 13 17 19 23 29 31 37 41 43 47
```

52)

```
      for i in range(1, 31):
    if i <= 10 or i >= 20:
    print(i)
```

53) def is_palindrome(s):

```
    s = s.lower().replace(" ", "")  # convert to lowercase and remove spaces
    return s == s[::-1]  # compare with the reverse of the string

# Example usage
print(is_palindrome("Madam"))  # True
print(is_palindrome("wow"))  # True
print(is_palindrome("cycle"))  # False
```

```
54) def print_pattern(n):
    for i in range(n, 0, -1):
        for j in range(i, 0, -1):
            print("*", end="")
        for k in range(n-i+1):
            print(" ", end="")
        for l in range(i, 0, -1):
            print("*", end="")
        print()

print_pattern(4)
print_pattern(5)
```

```
55) string = "Codoid innovations"
vowels = "aeiouAEIOU"
count = 0

for i in string:
    if i in vowels:
        count += 1
        print(i)

print("Total number of vowels in the given string:", count)
```

```
56) arr = [5, 4, 10, 20, 4, 6, 10, 39, 4, 39]
result = []
for num in arr:
 if num not in result:
  result.append(num)
print(result)
```

```
57) def find_largest(arr):
    max_num = arr[0]
    for num in arr:
        if num > max_num:
            max_num = num
    return max_num

# example usage
array = [5, 10, 15, 20, 25]
largest_num = find_largest(array)
print("The largest number is:", largest_num)
```

```
58) def replace_vowels_with_s(string):
    vowels = ['a', 'e', 'i', 'o', 'u']
    result = ""
    for char in string:
        if char.lower() in vowels:
            result += 'S'
        else:
            result += char
    return result

# Example usage:
string = "Python Pythsn"
result = replace_vowels_with_s(string)
print(result)
```

# Sql task

1) SELECT UPPER('hello world');

In SQL, scalar functions are functions that take one or more input parameters and return a single value. They are used to perform computations or transformations on values within a query.

There are several types of scalar functions in SQL, such as string functions, math functions, and date functions.

Here is an example of a scalar function in SQL that converts a string to uppercase:

2) In relational databases, a join operation is used to combine rows from two or more tables based on a related column between them. The most common types of joins are:

1. INNER JOIN: Returns only the rows that have matching values in both tables.
2. LEFT JOIN (or LEFT OUTER JOIN): Returns all the rows from the left table and matching rows from the right table. If there is no matching row in the right table, NULL values are returned.
3. RIGHT JOIN (or RIGHT OUTER JOIN): Returns all the rows from the right table and matching rows from the left table. If there is no matching row in the left table, NULL values are returned.
4. FULL JOIN (or FULL OUTER JOIN): Returns all the rows from both tables. If there is no matching row in either table, NULL values are returned.
5. SELECT employees.name, departments.name
6. FROM employees
7. FULL OUTER JOIN departments
8. ON employees.department_id

3) In this example, replace mytable with the name of your table, old_column_name with the current name of the column you want to rename, and new_column_name with the new name you want to give the column.

For instance, let's assume you have a table called employees with a column called emp_name and you want to rename the column to employee_name. Here's the SQL query to rename the column:

```
ALTER TABLE mytable

RENAME COLUMN old_column_name TO new_column_name;
```

4)

```
SELECT column1, column2, COUNT(*)

FROM table_name

GROUP BY column1, column2

HAVING COUNT(*) > 1;
```

5)

The DISTINCT keyword is used in SQL to retrieve only unique/distinct values from a table or result set. It eliminates all the duplicate rows and displays only one instance of each unique row.

```
SELECT DISTINCT column_name1, column_name2, ...
FROM table_name;
```

```
6) DELETE FROM students

WHERE id NOT IN

  (SELECT MIN(id)

   FROM students

   GROUP BY name, age)
```

```
7)
 SELECT MAX(salary) AS max_salary
FROM employee
WHERE department = 'department_name';
```

```
8)
  SELECT salary * 1.1 AS increased_salary FROM employee;
SELECT * FROM employee WHERE salary > 50000;
SELECT * FROM employee WHERE salary > 50000 AND department =
'Sales';
UPDATE employee SET salary = salary + 10000 WHERE department =
'Marketing';
SELECT first_name || ' ' || last_name AS full_name FROM employee;
```

```
9)
SELECT TOP 5 * FROM Employee;

10) SELECT * FROM Employee ORDER BY id DESC LIMIT 5;
SELECT * FROM
    (SELECT * FROM Employee ORDER BY id DESC)
WHERE ROWNUM <= 5;
```

```
11) SELECT salary
FROM (
  SELECT salary, RANK() OVER (ORDER BY salary DESC) as rank
  FROM Employee
) as emp_rank
WHERE rank = 3;
```

12)

```
    CREATE TABLE new_employee AS
SELECT * FROM employee;
```

13)

```
SELECT EmployeeID, EmployeeName, Salary
FROM Employee
LEFT JOIN Department ON
Employee.DepartmentID=Department.DepartmentID
WHERE Department.DepartmentID IS NULL;
```

Here, we are selecting the EmployeeID, EmployeeName, and Salary columns from the Employee table, and joining it with the Department table on the DepartmentID column. Then we are filtering out the records where the DepartmentID is NULL, which means that the Employee is not assigned to any department.