**LAB CYCLE 4**

**PROGRAM 1**

AIM

write a program to print the fibonacci series using recursion

PSEUDOCODE

SOURCE CODE

def fibonacci(n):

if n<=1:

return n

else:

return fibonacci(n-1)+fibonacci(n-2)

num\_terms=int(input("enter the number of terms :"))

if num\_terms<=0:

print("please enter a positive integer")

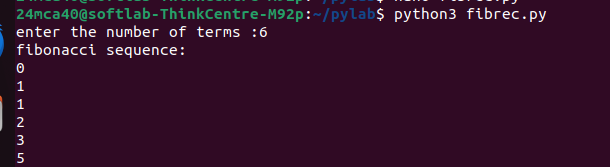
else:

print("fibonacci sequence:")

for i in range(num\_terms):

print(fibonacci(i))

OUTPUT



**PROGRAM 2**

AIM

Write the program to implement a menu-driven calculator. Use separate functions for the

different operations

PSEUDO CODE

SOURCE CODE

def add(x, y):

return x + y

def subtract(x, y):

return x - y

def multiply(x, y):

return x \* y

def divide(x, y):

if y == 0:

return "Error! Division by zero."

return x / y

while True:

print("\nSelect an operation:")

print("1. Addition")

print("2. Subtraction")

print("3. Multiplication")

print("4. Division")

print("5. Exit")

choice = input("Enter choice (1/2/3/4/5): ")

if choice == '5':

print("Exiting ")

break

if choice in ('1', '2', '3', '4'):

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

if choice == '1':

print(f"Result: {num1} + {num2} = {add(num1, num2)}")

elif choice == '2':

print(f"Result: {num1} - {num2} = {subtract(num1, num2)}")

elif choice == '3':

print(f"Result: {num1} \* {num2} = {multiply(num1, num2)}")

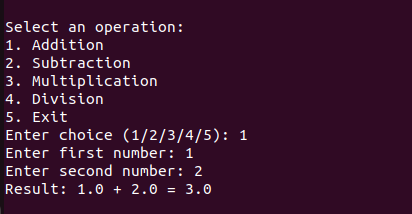
elif choice == '4':

print(f"Result: {num1} / {num2} = {divide(num1, num2)}")

else:

print("Invalid choice! Please select a valid option.")

OUTPUT



**PROGRAM 3**

AIM

Write a program to print the nth prime number.

[Use function to check whether a number is prime or not

PSEUDO CODE

SOURCE CODE

def is\_prime(num):

if num <= 1:

return False

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

return False

return True

def nth\_prime(n):

count = 0

num = 2

while True:

if is\_prime(num):

count += 1

if count == n:

return num

num +=1

n = int(input("Enter the position of the prime number you want to find (n): "))

if n <= 0:

print("Please enter a positive integer greater than 0.")

else:

print(f"The {n}-th prime number is: {nth\_prime(n)}")

OUTPUT



**PROGRAM 4**

AIM

Write lambda functions to find the area of square, rectangle and triangle.

PSEUDO CODE

SOURCE CODE

area\_square = lambda side: side \* side

area\_rectangle = lambda length, width: length \* width

area\_triangle = lambda base, height: 0.5 \* base \* height

while True:

print("\nChoose a shape to calculate the area:")

print("1. Square")

print("2. Rectangle")

print("3. Triangle")

print("4. Exit")

choice = input("Enter choice (1/2/3/4): ")

if choice == '1':

side = float(input("Enter the side length of the square: "))

print(f"The area of the square is: {area\_square(side)}")

elif choice == '2':

length = float(input("Enter the length of the rectangle: "))

width = float(input("Enter the width of the rectangle: "))

print(f"The area of the rectangle is: {area\_rectangle(length, width)}")

elif choice == '3':

base = float(input("Enter the base length of the triangle: "))

height = float(input("Enter the height of the triangle: "))

print(f"The area of the triangle is: {area\_triangle(base, height)}")

elif choice == '4':

print("Exiting the program. Goodbye!")

break

else:

print("Invalid choice! Please select a valid option.")

OUTPUT



**PROGRAM 5**

AIM

Write a program to display powers of 2 using anonymous function. [ Hint use map and

lambda function)

PSEUDO CODE

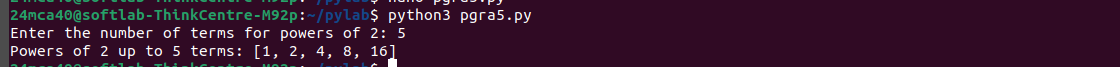
SOURCE CODE

num\_terms = int(input("Enter the number of terms for powers of 2: "))

powers\_of\_2 = list(map(lambda x: 2 \*\* x, range(num\_terms)))

print(f"Powers of 2 up to {num\_terms} terms: {powers\_of\_2}")

OUTPUT



**PROGRAM 6**

AIM

Write a program to sum the series 1/1! + 4/2! + 27/3! + ….. + nth term. [ Hint

Use a function to find the factorial of a number].

PSEUDO CODE

SOURCE CODE

def factorial(num):

if num == 0 or num == 1:

return 1

else:

result = 1

for i in range(2, num + 1):

result \*= i

return result

def sum\_series(n):

series\_sum = 0

for i in range(1, n + 1):

term = (i \*\* i) / factorial(i)

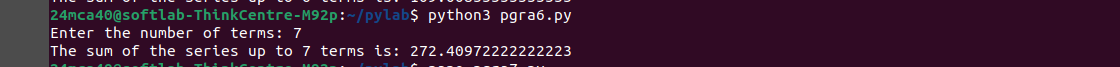
series\_sum += term

return series\_sum

n = int(input("Enter the number of terms: "))

print(f"The sum of the series up to {n} terms is: {sum\_series(n)}")

OUTPUT



**PROGRAM 7**

AIM

Write a function called compare which takes two strings S1 and S2 and an integer n as

arguments. The function should return True if the first n characters of both the strings

are the same else the function should return False

PSEUDO CODE

SOURCE CODE

def compare(S1, S2, n):

if S1[:n] == S2[:n]:

return True

else:

return False

S1 = input("Enter the first string (S1): ")

S2 = input("Enter the second string (S2): ")

n = int(input("Enter the number of characters to compare: "))

result = compare(S1, S2, n)

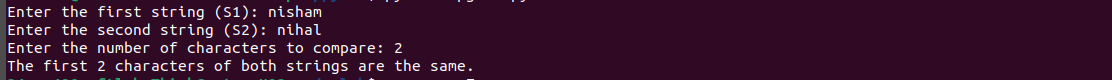
if result:

print("The first", n, "characters of both strings are the same.")

else:

print("The first", n, "characters of both strings are not the same.")

OUTPUT



**PROGRAM 8**

AIM

Write a program using functions to implement these formulae for permutations and

combinations.

The Number of permutations of n objects taken r at a time: p(n, r) = n!/(n − r)!.

The Number of combinations of n objects taken r at a time is: c(n, r) = n!/(r! ∗ (n − r)!)

PSEUDO CODE

SOURCE CODE

def factorial(num):

if num == 0 or num == 1:

return 1

else:

result = 1

for i in range(2, num + 1):

result \*= i

return result

def permutations(n, r):

return factorial(n) // factorial(n - r)

def combinations(n, r):

return factorial(n) // (factorial(r) \* factorial(n - r))

n = int(input("Enter the value of n (total objects): "))

r = int(input("Enter the value of r (objects taken at a time): "))

if r > n:

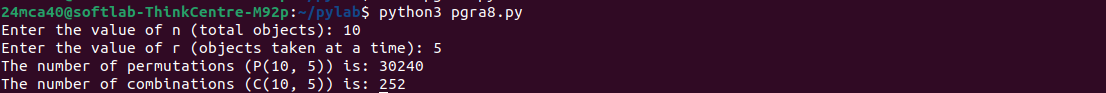
print("Invalid input: r should be less than or equal to n.")

else:

print(f"The number of permutations (P({n}, {r})) is: {permutations(n, r)}")

print(f"The number of combinations (C({n}, {r})) is: {combinations(n, r)}")

OUTPUT



**PROGRAM 9**

**AIM**

Write a program to add variable length integer arguments passed to the function. [Also

demo the use of docstrings]

**SOURCE CODE**

def add\_numbers(\*args):

"""

Adds a variable number of integer arguments.

parameters:

\*args:A variable length list of Integers to be added.

returns:

int:the sum of all the integers passed as argumens.

"""

if not all(isinstance(arg,int)for arg in args):

raise valueError("All arguments must be integers!!")

return sum(args)

print("sum of 1,2,3:",add\_numbers(1,2,3))

print("sum of 10,20,30,40:",add\_numbers(10,20,30,40))

**OUTPUT**

****

**PROGRAM 10**

**AIM**

Write a program to display multiples of 3 using anonymous function. [ Hint use filter and

lambda function)

**SOURCE CODE**

terms = int(input("Enter the number of terms: "))

indices = range(terms)

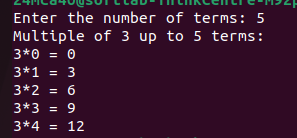
Multiple\_of\_3 = list(filter(lambda x: True, map(lambda i: 3\*i, indices)))

print(f"Multiple of 3 up to {terms} terms:")

for i in range(terms):

print(f"3\*{i} = {Multiple\_of\_3[i]}")

**OUTPUT**

****