**LAB PROGRAM -1**

R.KISHOR KUMAR

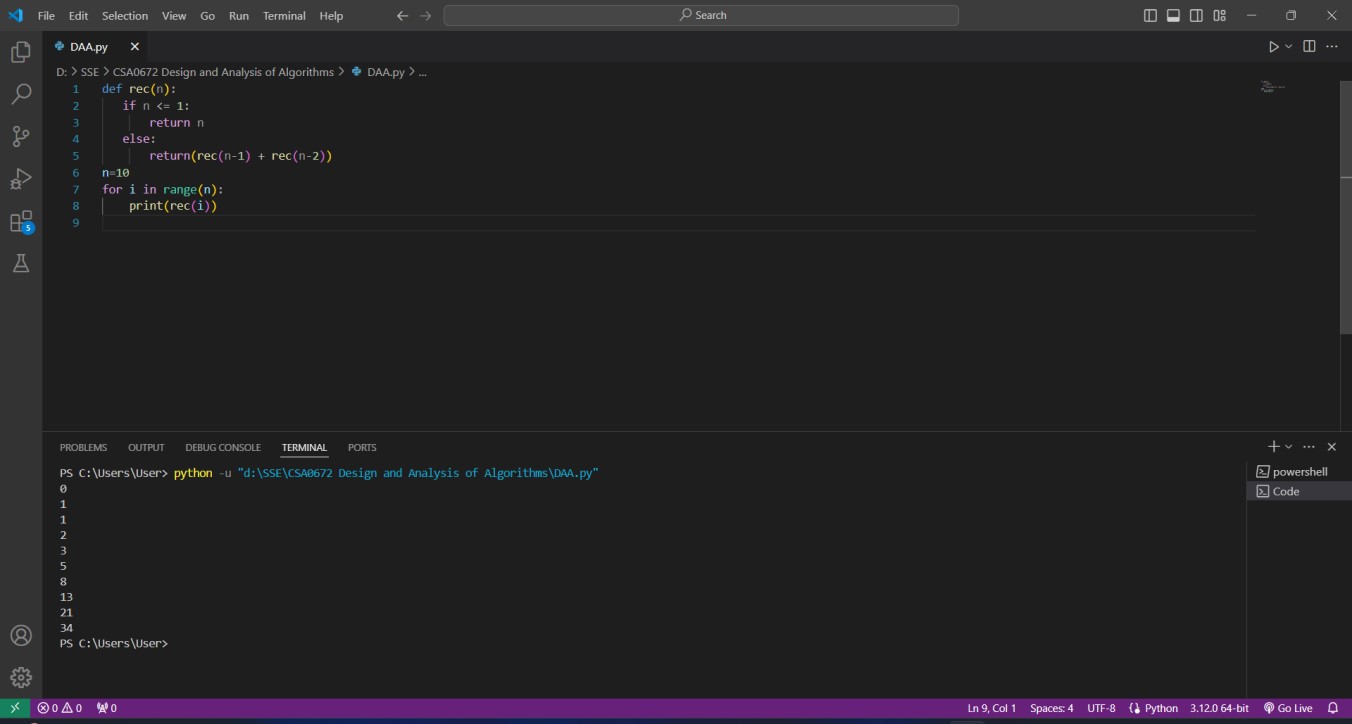
192311022

1. Write a program to Print Fibonacci Series using recursion.

Code:

def rec(n): if n <= 1: return n else: return(rec(n-1) + rec(n-2)) n=10 for i in range(n): print(rec(i))

**Screenshot for I/O:**



Time Complexity:O(n)

2. Write a program to check the given no is Armstrong or not using recursive function.

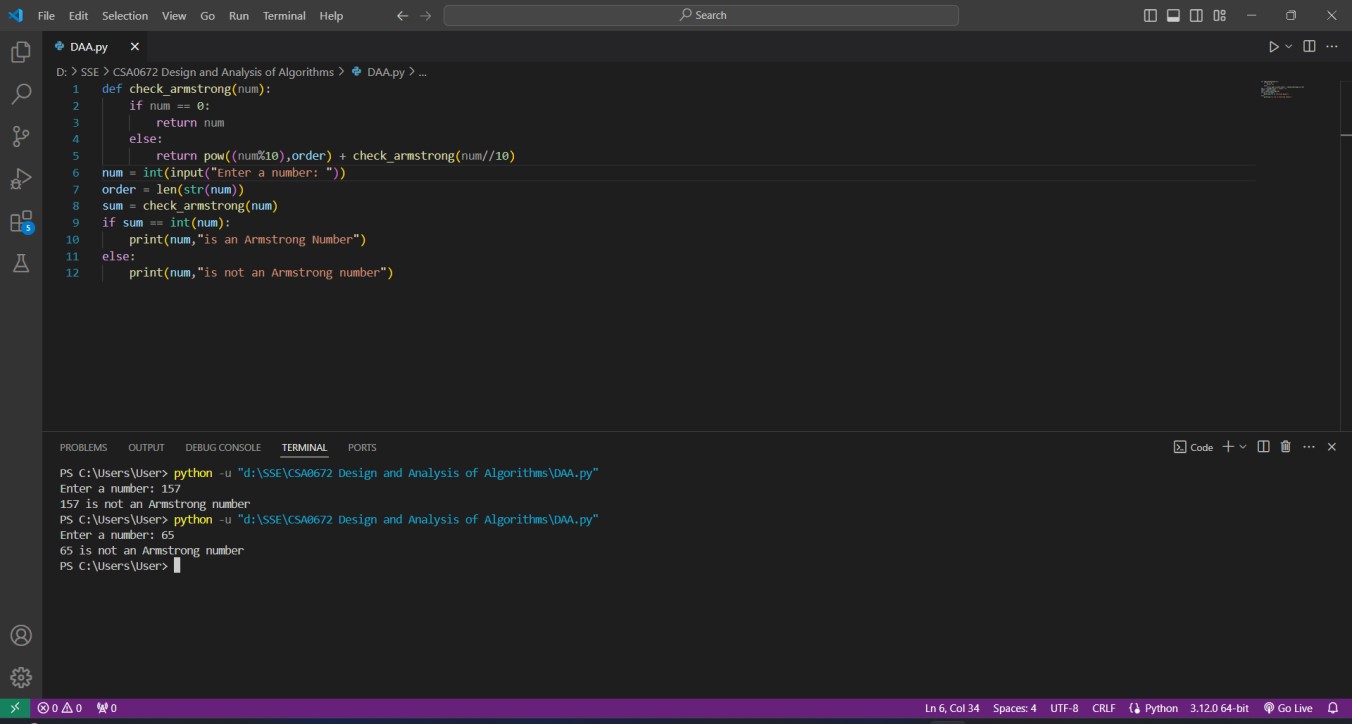
Code:

def check\_armstrong(num): if num == 0: return num else:

return pow((num%10),order) + check\_armstrong(num//10) num = int(input("Enter a number: ")) order = len(str(num)) sum = check\_armstrong(num) if sum == int(num): print(num,"is an Armstrong Number") else:

print(num,"is not an Armstrong number")

**Screenshot for I/O:**



**Time Complexity:O(n)**

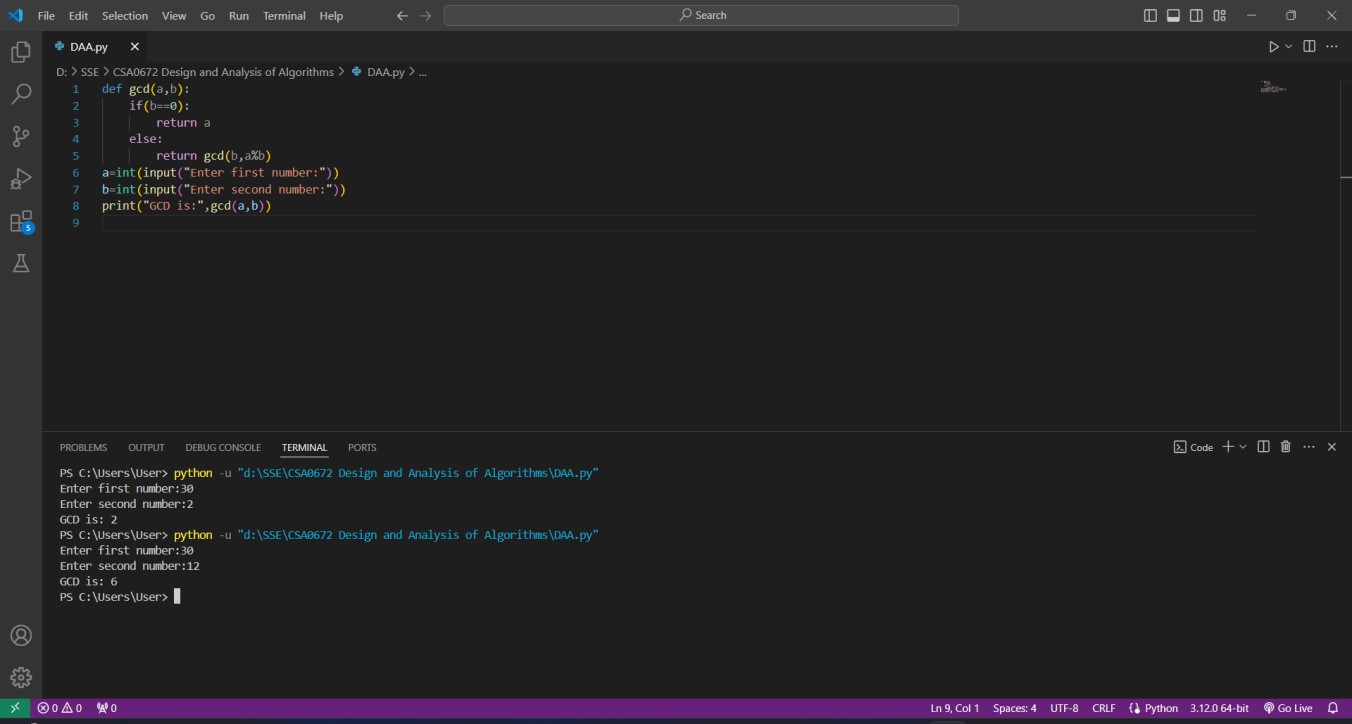
1. Write a program to find the GCD of two numbers using recursive factorization Code:

def gcd(a,b): if(b==0): return a else:

return gcd(b,a%b) a=int(input("Enter first number:")) b=int(input("Enter second number:"))

print("GCD is:",gcd(a,b))

**Screenshot for I/O:**



**Time Complexity: O(n)**

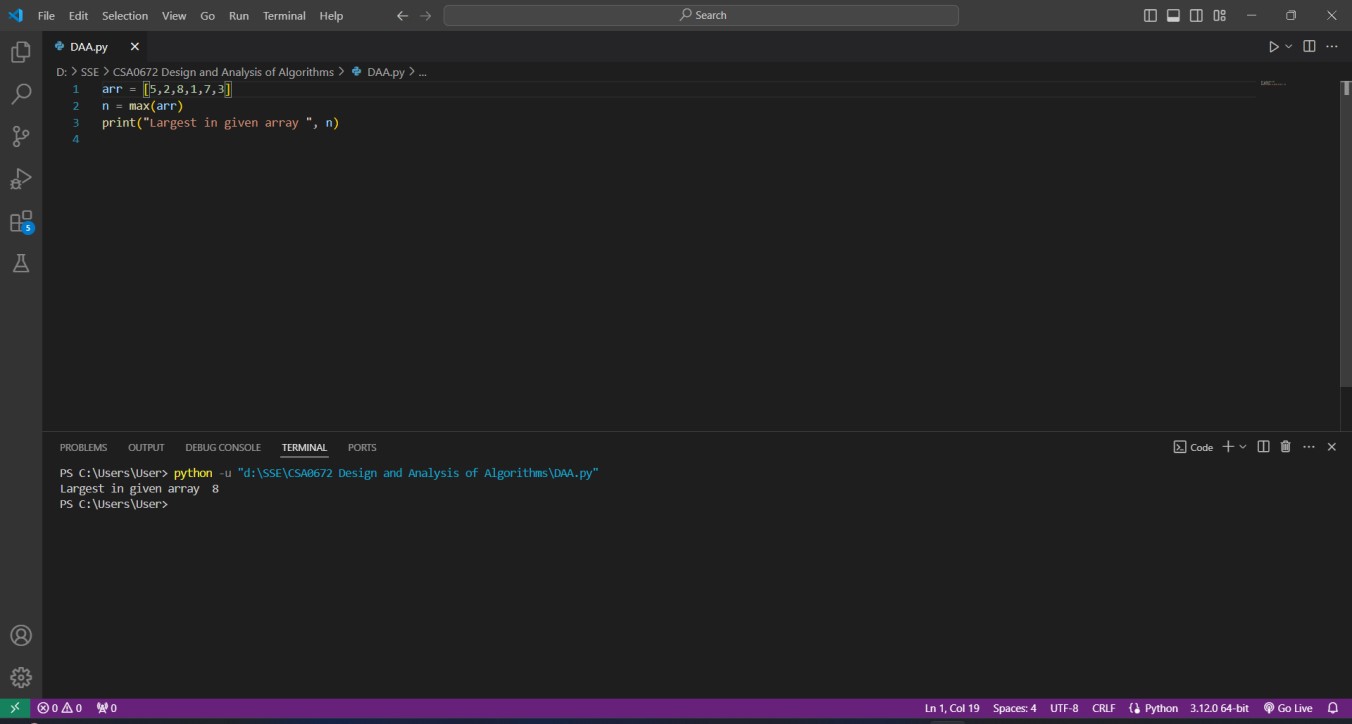
1. Write a program to get the largest element of an array.

Code:

arr = [5,2,8,1,7,3] n = max(arr)

print("Largest in given array ", n)

**Screenshot for I/O:**



**Time Complexity: O(n)**

1. Write a program to find the Factorial of a number using recursion.

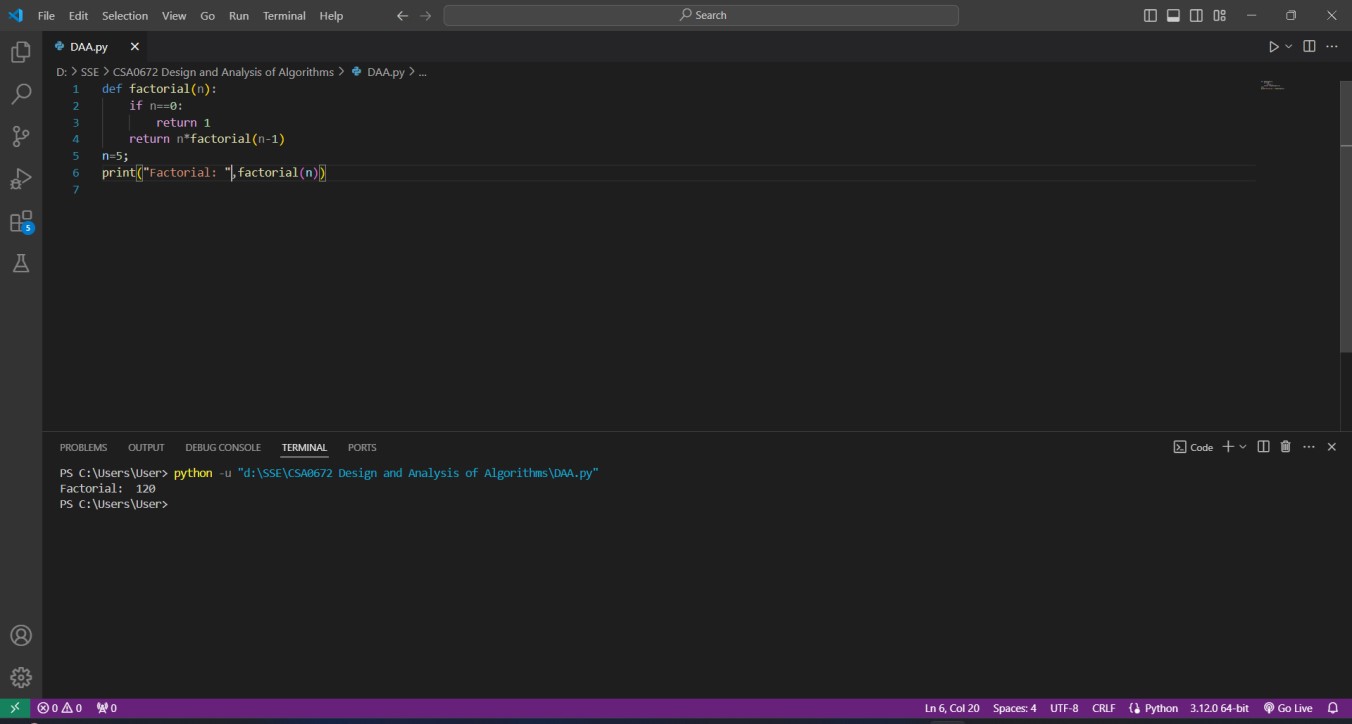
Code:

def factorial(n): if n==0: return 1

return n\*factorial(n-1) n=5;

print("Factorial: ",factorial(n))

**Screenshot for I/O:**



**Time Complexity: O(n)**

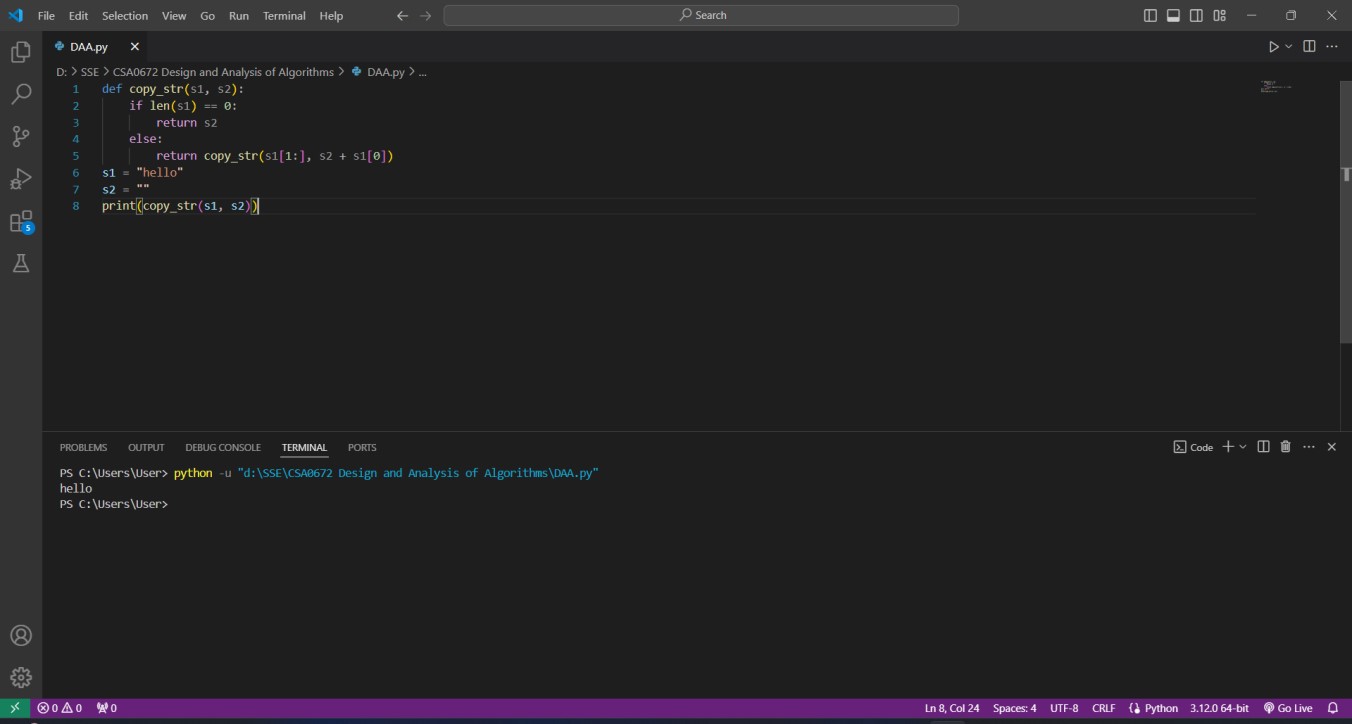
1. Write a program for to copy one string to another using recursion Code:

def copy\_str(s1, s2): if len(s1) == 0: return s2 else:

return copy\_str(s1[1:], s2 + s1[0]) s1 = "hello" s2 = ""

print(copy\_str(s1, s2))

Screenshot for I/O:



**Time Complexity: O(n)**

1. Write a program to print the reverse of a string using recursion Code:

def reverse(s): if len(s) == 0:

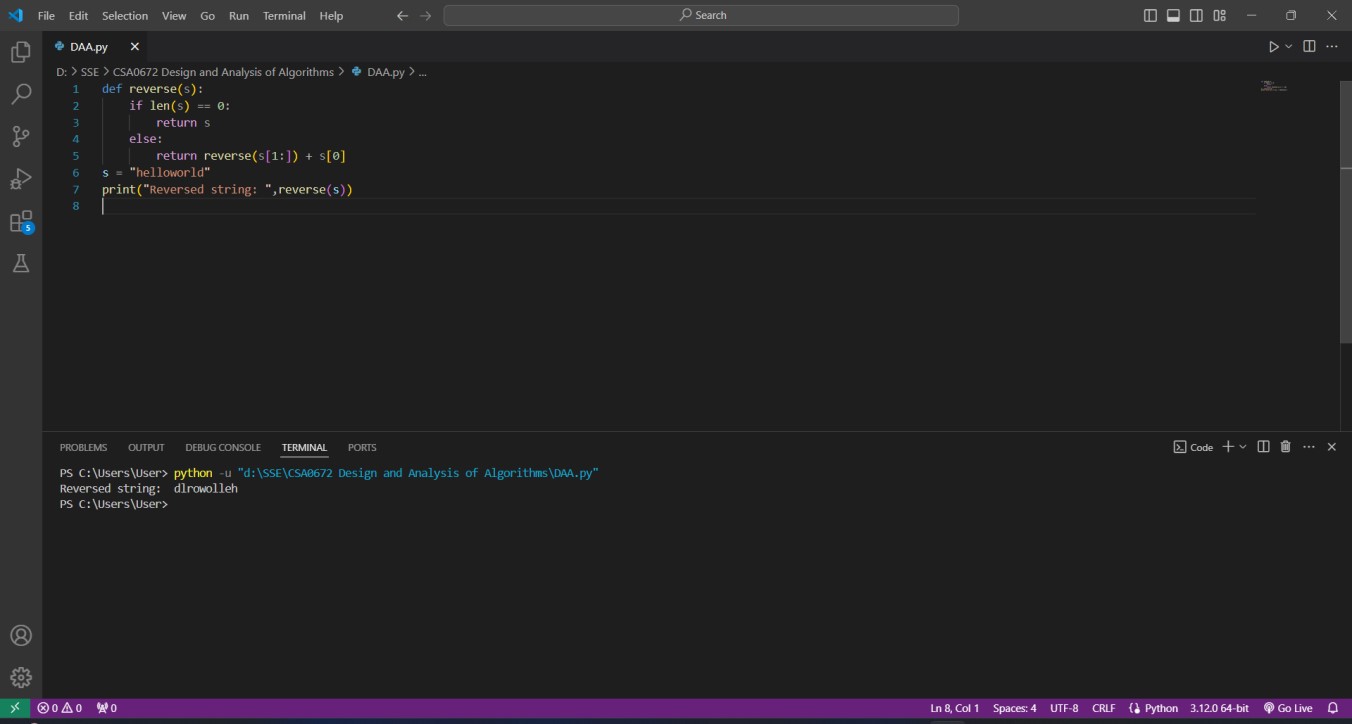
return s else:

return reverse(s[1:]) + s[0]

s = "helloworld"

print("Reversed string: ",reverse(s))

**Screenshot for I/O:**



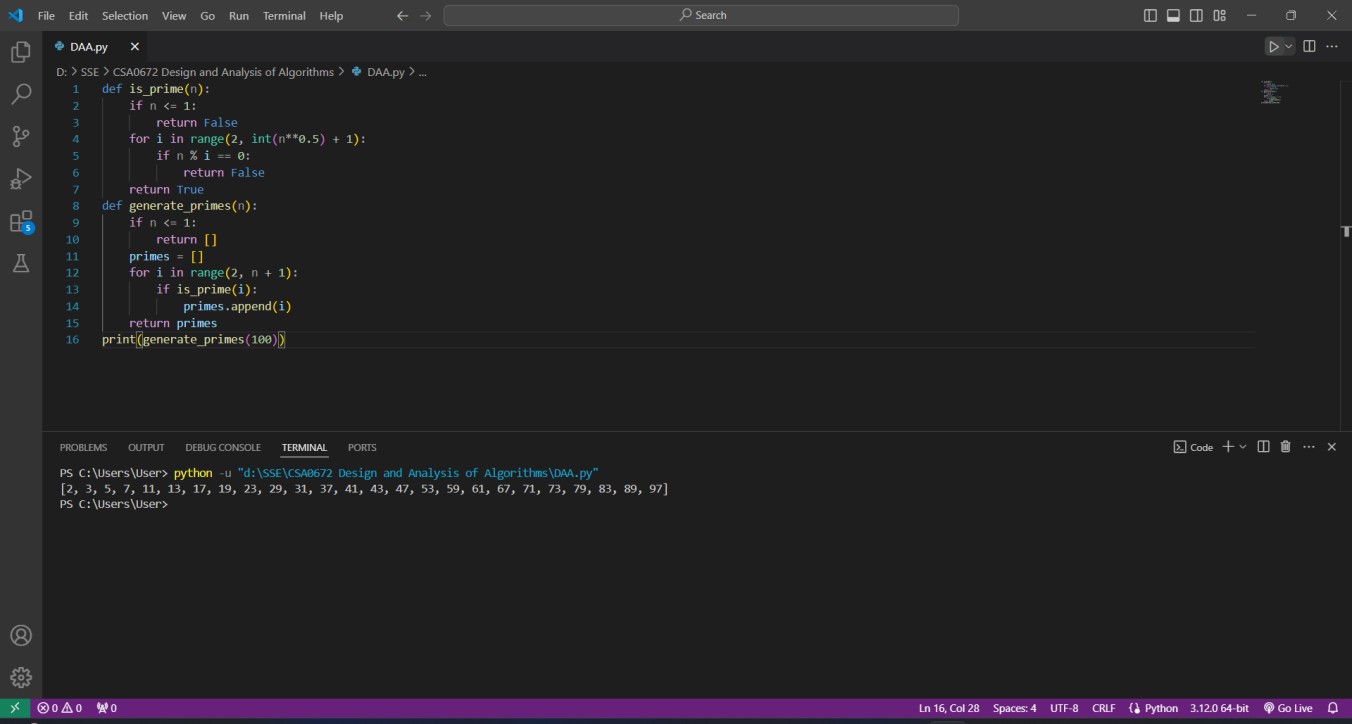
**Time Complexity: O(n)**

8. Write a program to generate all the prime numbers using recursion Code:

def is\_prime(n): if n <= 1: return False for i in range(2, int(n\*\*0.5) + 1): if n % i == 0: return False return True def generate\_primes(n): if n <= 1: return [] primes = [] for i in range(2, n + 1): if is\_prime(i): primes.append(i) return primes

print(generate\_primes(100))

**Screenshot for I/O:**



**Time Complexity: O(n\*m)**

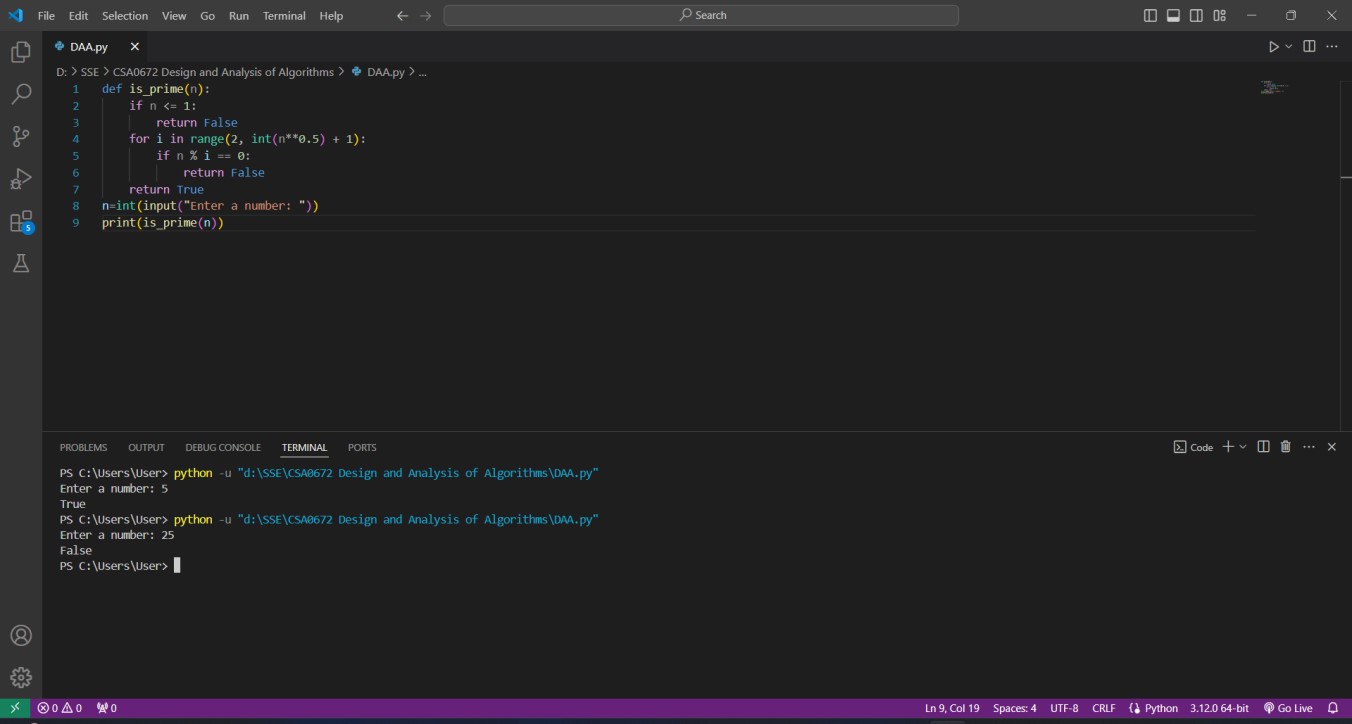
9. Write a program to check a number is a prime number or not using recursion.

Code:

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

n=int(input("Enter a number: ")) print(is\_prime(n))

**Screenshot for I/O:**



**Time Complexity: O(n)**

10. Write a program for to check whether a given String is Palindrome or not using recursion.

Code:

def is\_palindrome(s): if len(s) < 1: return True else: if s[0] == s[-1]:

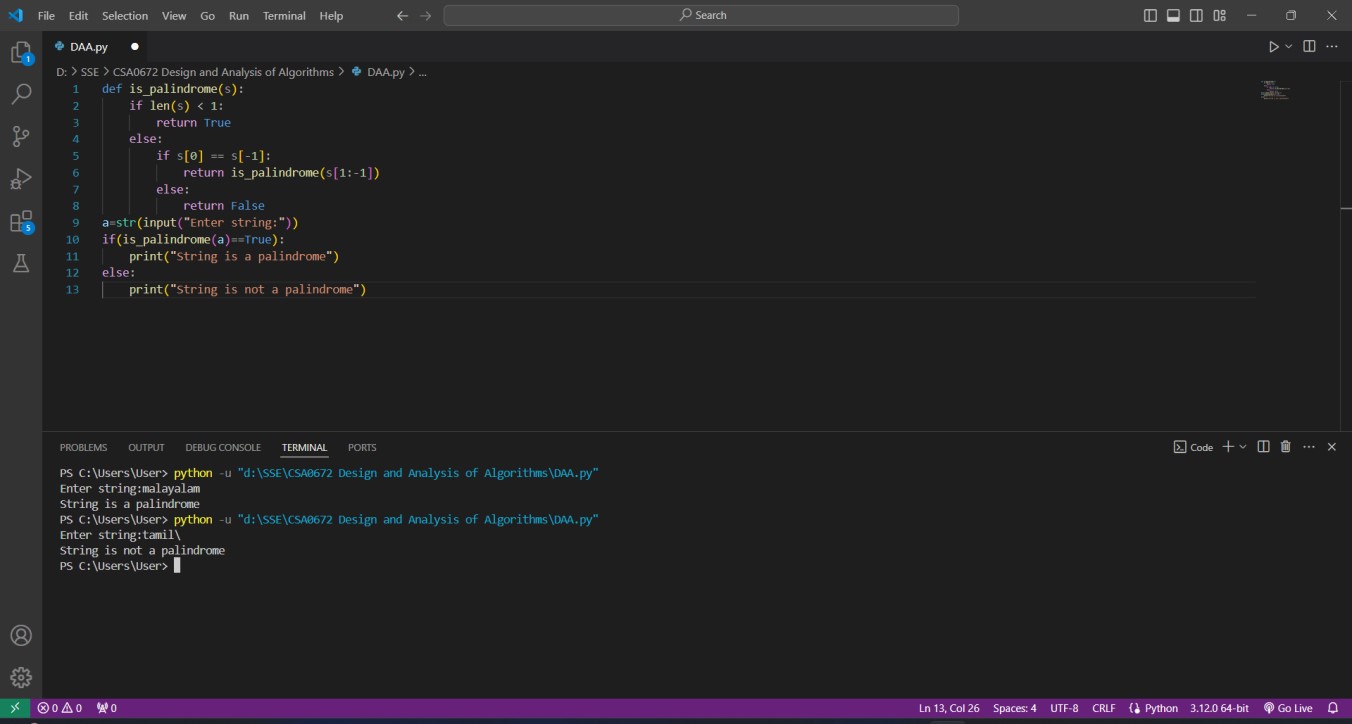
return is\_palindrome(s[1:-1]) else:

return False a=str(input("Enter string:")) if(is\_palindrome(a)==True):

print("String is a palindrome") else:

print("String is not a palindrome")

**Screenshot for I/O:**



**Time Complexity: O(n)**