DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

**S**ubject **N**ame:

**AI Decision Support**

**L**ab **N**umber:

**01**

**SUBMITTED BY:**

**Muhammad Hashir Ashraf** **Awan** reg# (**408557)**

DE- 44 Dept **CE**-**B**

**Submitted to:**

**Madam Eiman Fatima**

**Submission Date:**

11/09/2025

**Hardware/Software required:**

Hardware: **PC**

Software Tool: **Visual Studio Code**

**Task 1:**

Write a program that takes a sentence from the user and checks whether it is a pangram (a sentence containing every letter of the English alphabet at least once, ignoring case and spaces).  
Example: *“The quick brown fox jumps over the lazy dog”* → Pangram.

**Solution:**

a=str(input('Enter a string: '))

print(len(a))

range=set('abcdefghijklmnopqrstuvwxyz')

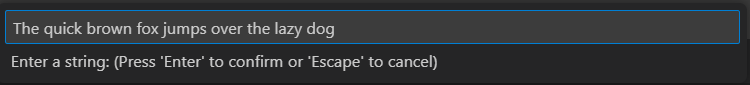
if range.issubset(a.lower()):

    print('String is a panagram')

else:

    print('String is not a panagram')

**Output:**

****

**A black background with white text

AI-generated content may be incorrect.**

**Task 2:**

Write a program that checks if a given number is a Perfect number. A Perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself).

*Example: 28 → 1 + 2 + 4 + 7 + 14 = 28 → Perfect number.*

**Solution:**

a=int(input('Enter a number:'))

divisors=[]

for i in range (1,(a//2)+1):

    if a%i==0:

        divisors.append(i)

if sum(divisors)==a:

    print('Perfect number')

else:

    print('Not a perfect number')

**Output:**

**A black and white text

AI-generated content may be incorrect.**

****

**Task 3:**

Write a program that generates random passwords containing both letters and digits. The user should enter the number of passwords to generate and their length. Each password must contain at least one digit and one letter. Display all the generated passwords.

**Solution:**

import random,string

num\_passwords = int(input("Number of passwords to generate: "))

length = int(input("Length of each password: "))

for i in range(1,num\_passwords+1):

    password = [

        random.choice(string.ascii\_letters),

        random.choice(string.digits)

    ]

    all\_chars = string.ascii\_letters + string.digits

    password += random.choices(all\_chars, k=length-2)

    password\_str = ''.join(password)

    print(password\_str)

**Output:**

**A black background with white text

AI-generated content may be incorrect.**

**Task 4:**

Write a program to create a class Circle with attribute radius. Include methods area() to compute the area of the circle and circumference() to calculate its perimeter. Take the radius as input from the user in the main program, create an object of the class, and display the circle’s area and circumference.

**Solution:**

class Circle:

    def \_\_init\_\_(self, radius):

        self.radius = radius

    def area(self):

        print ('Area is = ',3.14\* self.radius \* self.radius)

    def circum(self):

        print('Circumference is = ',2\*3.14\* self.radius)

if \_\_name\_\_ == "\_\_main\_\_":

    circle = Circle(int(input('Enter radius of circle: ')))

    circle.area()

    circle.circum()

**Output:**

**A black background with white text

AI-generated content may be incorrect.**

**Task 5:**

Create a Binary Search Tree (BST) and insert a series of integers into it. Write a program to search for the node 25 in the BST (as covered in your Data Structures and Algorithms course). You can hard-code the values or take them as input from the user. Also, state the time complexity of the search operation in Big-O notation

**Code:**

def create\_node(value):

    return {'value': value, 'left': None, 'right': None}

def insert(root, value):

    if root is None:

        return create\_node(value)

    if value < root['value']:

        root['left'] = insert(root['left'], value)

    elif value > root['value']:

        root['right'] = insert(root['right'], value)

    return root

def search(root, value):

    if root is None:

        return False

    if value == root['value']:

        return True

    elif value < root['value']:

        return search(root['left'], value)

    else:

        return search(root['right'], value)

values = [50, 30, 70, 20, 40, 60, 80, 25]

root = None

for v in values:

    root = insert(root, v)

search\_value = 25

if search(root, search\_value):

    print("Node found")

else:

    print("Node not found")

**Output:**

****