|  |  |  |
| --- | --- | --- |
|  | **THADOMAL SHAHANI ENGINEERING COLLEGE** |  |
| **DEPARTMENT OF INFORMATION TECHNOLOGY** |

**Roll no: I-62**

**4.File Handling – LO3**

**1)Aim:**

Develop a Python program that reads a text file and prints words of specified lengths (e.g., three, four, five, etc.) found within the file.

**Theory:**

* Python allows you to efficiently read and write files.
* String manipulation functions let you filter words by length.
* Iterating through file content enables precise word extraction.
* Regular expressions strengthen pattern-matching capabilities.
* Handling exceptions like FileNotFoundError ensures smooth execution.

**Program :**

with open("names.txt", "r") as file:

text = file.read()

words = text.split()

desired\_length = int(input("Enter the desired word length: "))

words\_found = [word for word in words if len(word) == desired\_length]

if words\_found:

print(f"Words of length {desired\_length}:")

print(", ".join(words\_found))

else:

print(f"No words of length {desired\_length} found.")

**Output:**

Enter the desired word length: 4

Words of length 4:

done, Kuhu

Enter the desired word length: 5

Words of length 5:

Ankit, Manav, Three

Enter the desired word length: 6

Words of length 6:

Aniket

Enter the desired word length: 7

Words of length 7:

Dishita, Shaurya

Enter the desired word length: 8

Words of length 8:

Vanshita, Aryaveer

Enter the desired word length: 9

No words of length 9 found.

**Conclusion:**

File handling and string manipulation enable efficient extraction of words based on length from text files**.**

**2)Aim:**

Finding Closest Points in 3D Coordinates from CSV: Write a python code to take a csv file as input with coordinates of points in three dimensions. Find out the two closest points.

**Program:**

import csv

import math

def calculate\_distance(coords1, coords2):

# Calculate Euclidean distance between two 3D points

return math.sqrt(

(int(coords1[0]) - int(coords2[0])) \*\* 2 +

(int(coords1[1]) - int(coords2[1])) \*\* 2 +

(int(coords1[2]) - int(coords2[2])) \*\* 2

)

# Open and process the CSV file

with open("file.csv", mode="r") as file:

reader = csv.reader(file)

my\_list = [row for row in reader] # Read all rows into a list

print("Coordinates List:", my\_list)

if len(my\_list) < 2:

print("Not enough data points to calculate a distance.")

else:

min\_distance = float("inf") # Initialize with a very large value

min\_coords = None

for i in range(len(my\_list)):

for j in range(i + 1, len(my\_list)): # Only compare unique pairs

coord1 = my\_list[i]

coord2 = my\_list[j]

distance = calculate\_distance(coord1, coord2)

if distance < min\_distance:

min\_distance = distance

min\_coords = (coord1, coord2)

print(f"Minimum distance is {min\_distance} between {min\_coords[0]} and {min\_coords[1]}")

**Output:**

Coordinates List: [['1', '2', '3'], ['4', '5', '6'], ['7', '8', '9'], ['2', '3', '4']]

Minimum distance is 1.7320508075688772 between ['1', '2', '3'] and ['2', '3', '4']

**Conclusion:**

The combination of CSV file processing and mathematical calculations ensures precise identification of the nearest points in 3D space**.**

**3)Aim:**

Sorting City Names from File: Write a python code to take a file which contains city names on each line. Alphabetically sort the city names and write it in another file.

**Theory:**

* Processing file content line by line enables efficient management of city names.
* Utilizing sorting methods, such as `sorted()` or `.sort()`, arranges the data in alphabetical order.
* Saving the sorted data into a separate file ensures the original data remains unaltered.
* Ensuring proper file encoding avoids potential character-related complications.
* Eliminating duplicate entries improves the overall reliability and precision of the data.

**Program:**

with open("cities.txt", "r") as file:

data = file.read()

cities = [city.strip() for city in data.split("\n") if city.strip()]

cities.sort()

for city in cities:

print(city)

**Output:**

Bengaluru

Chennai

Delhi

Hyderabad

Jaipur

Kolkata

Mumbai

Pune

**Conclusion:**

Reading and sorting files facilitate the organized structuring of city names, enhancing the efficiency and reliability of data management processes.