Module VIII

# Program to create a dictionary with alphabets as keys and occurrences as values

input\_string = input("Enter a string: ")

# Initialize an empty dictionary

char\_count = {}

# Iterate through each character in the string

for char in input\_string:

if char.isalpha(): # Check if the character is an alphabet

char = char.lower() # Convert to lowercase for uniformity

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

print("Character count dictionary:", char\_count)

# Program to enter names and marks, and print names according to marks in descending order

num\_students = int(input("Enter the number of students: "))

student\_marks = {}

# Input student names and their marks

for \_ in range(num\_students):

name = input("Enter student name: ")

marks = int(input("Enter student marks: "))

student\_marks[name] = marks

# Sort the dictionary by marks in descending order

sorted\_students = dict(sorted(student\_marks.items(), key=lambda item: item[1], reverse=True))

print("Students sorted by marks in descending order:")

for name, marks in sorted\_students.items():

print(f"{name}: {marks}")

# Program to manage a dictionary with name as key and date of birth as value

def display\_menu():

print("\nMenu:")

print("1. Insert new entry")

print("2. Delete an entry")

print("3. Update an entry")

print("4. Display all entries")

print("5. Exit")

def insert\_entry(birthday\_dict):

name = input("Enter name: ")

dob = input("Enter date of birth (YYYY-MM-DD): ")

birthday\_dict[name] = dob

print("Entry added successfully.")

def delete\_entry(birthday\_dict):

name = input("Enter name to delete: ")

if name in birthday\_dict:

del birthday\_dict[name]

print("Entry deleted successfully.")

else:

print("Name not found.")

def update\_entry(birthday\_dict):

name = input("Enter name to update: ")

if name in birthday\_dict:

dob = input("Enter new date of birth (YYYY-MM-DD): ")

birthday\_dict[name] = dob

print("Entry updated successfully.")

else:

print("Name not found.")

def display\_entries(birthday\_dict):

print("\nAll entries:")

for name, dob in birthday\_dict.items():

print(f"{name}: {dob}")

# Main program

birthday\_dict = {}

while True:

display\_menu()

choice = input("Enter your choice: ")

if choice == '1':

insert\_entry(birthday\_dict)

elif choice == '2':

delete\_entry(birthday\_dict)

elif choice == '3':

update\_entry(birthday\_dict)

elif choice == '4':

display\_entries(birthday\_dict)

elif choice == '5':

print("Exiting the program.")

break

else:

print("Invalid choice. Please try again.")

Module IX

# Program to check if the string contains all the alphabets

def has\_all\_alphabets(input\_string):

alphabet\_set = set("abcdefghijklmnopqrstuvwxyz")

input\_set = set(input\_string.lower())

return alphabet\_set.issubset(input\_set)

input\_string = "the quick brown fox jumps over the lazy dog"

print("String contains all the alphabets:", has\_all\_alphabets(input\_string))

# Program to find the number of lowercase and uppercase characters in a string

def count\_case\_characters(input\_string):

lowercase\_count = sum(1 for char in input\_string if char.islower())

uppercase\_count = sum(1 for char in input\_string if char.isupper())

return lowercase\_count, uppercase\_count

input\_string = input("Enter a string: ")

lowercase\_count, uppercase\_count = count\_case\_characters(input\_string)

print("Number of lowercase characters:", lowercase\_count)

print("Number of uppercase characters:", uppercase\_count)

# Program to check if a string is a palindrome without using string methods

def is\_palindrome(input\_string):

left = 0

right = len(input\_string) - 1

while left < right:

if input\_string[left] != input\_string[right]:

return False

left += 1

right -= 1

return True

input\_string = input("Enter a string: ")

print("String is a palindrome:", is\_palindrome(input\_string))

Module X

# Program to show the use of local and global variables and lambda function

x = 10 # Global variable

def local\_global\_example():

x = 5 # Local variable

print("Local variable x:", x)

return x

def global\_example():

global x

x = 20 # Modify global variable

print("Modified global variable x:", x)

return x

# Lambda function to add two numbers

add = lambda a, b: a + b

print("Global variable x initially:", x)

local\_value = local\_global\_example()

global\_value = global\_example()

print("Lambda function add(3, 4):", add(3, 4))

# Program to perform linear search in a list

def linear\_search(lst, target):

for index, value in enumerate(lst):

if value == target:

return index

return -1

lst = [10, 20, 30, 40, 50]

target = int(input("Enter the target value: "))

index = linear\_search(lst, target)

if index != -1:

print(f"Target found at index {index}")

else:

print("Target not found in the list")

# Program to perform bubble sort in a list

def bubble\_sort(lst):

n = len(lst)

for i in range(n):

for j in range(0, n - i - 1):

if lst[j] > lst[j + 1]:

lst[j], lst[j + 1] = lst[j + 1], lst[j]

return lst

lst = [64, 34, 25, 12, 22, 11, 90]

sorted\_list = bubble\_sort(lst)

print("Sorted list:", sorted\_list)

# Program to sum the positive numbers, passing them as arguments

def sum\_positive\_numbers(\*args):

return sum(arg for arg in args if arg > 0)

# Example usage

print("Sum of positive numbers:", sum\_positive\_numbers(1, -2, 3, 4, -5))

Module XI

import math

def find\_square\_root():

try:

num = float(input("Enter a number: "))

if num < 0:

raise ValueError("Cannot compute square root of a negative number.")

sqrt = math.sqrt(num)

print(f"The square root of {num} is {sqrt}")

except ValueError as ve:

print(ve)

except Exception as e:

print("An unexpected error occurred:", e)

find\_square\_root()

def search\_by\_index():

try:

lst = [10, 20, 30, 40, 50]

index = int(input("Enter the index to search: "))

value = lst[index]

print(f"Value at index {index} is {value}")

except IndexError:

print("Error: Index is out of range.")

except ValueError:

print("Error: Invalid input. Please enter an integer.")

except Exception as e:

print("An unexpected error occurred:", e)

search\_by\_index()

Module XII

# Program to perform read-write operations on a file

file\_path = "example.txt"

# Writing to the file

with open(file\_path, 'w') as file:

file.write("Hello, World!\n")

file.write("This is a file handling example.\n")

# Reading from the file

with open(file\_path, 'r') as file:

content = file.read()

print("File content:")

print(content)

# Program to copy the content of one file to another file

source\_file\_path = "source.txt"

destination\_file\_path = "destination.txt"

# Copy content from source file to destination file

with open(source\_file\_path, 'r') as source\_file:

with open(destination\_file\_path, 'w') as destination\_file:

for line in source\_file:

destination\_file.write(line)

print("Content copied successfully from source to destination.")

# Program to count number of lines and words in a file

file\_path = "example.txt"

line\_count = 0

word\_count = 0

with open(file\_path, 'r') as file:

for line in file:

line\_count += 1

words = line.split()

word\_count += len(words)

print(f"Number of lines: {line\_count}")

print(f"Number of words: {word\_count}")

# Program to print frequency of each character in a file

file\_path = "example.txt"

char\_count = {}

with open(file\_path, 'r') as file:

for line in file:

for char in line:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

print("Character frequency in file:")

for char, count in char\_count.items():

print(f"{char}: {count}")

Module XIII

# Program to create a class Student with attributes name, age, and grade

class Student:

def \_\_init\_\_(self, name, age, grade):

self.name = name

self.age = age

self.grade = grade

def print\_info(self):

print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}")

# Creating three Student objects

student1 = Student("Alice", 20, "A")

student2 = Student("Bob", 21, "B")

student3 = Student("Charlie", 22, "A")

# Printing the information of the students

student1.print\_info()

student2.print\_info()

student3.print\_info()

# Program to demonstrate inheritance with CSEStudent class

class Student:

def \_\_init\_\_(self, name, age, grade):

self.name = name

self.age = age

self.grade = grade

def print\_info(self):

print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}")

class CSEStudent(Student):

def \_\_init\_\_(self, name, age, grade, graduation\_year):

super().\_\_init\_\_(name, age, grade)

self.graduation\_year = graduation\_year

def print\_info(self):

print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}, Graduation Year: {self.graduation\_year}")

# Creating a CSEStudent object

cse\_student = CSEStudent("David", 23, "A", 2023)

# Printing the information of the CSEStudent

cse\_student.print\_info()

# Program to demonstrate polymorphism with Student and CSEStudent classes

class Student:

def \_\_init\_\_(self, name, age, grade):

self.name = name

self.age = age

self.grade = grade

def print\_info(self):

print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}")

class CSEStudent(Student):

def \_\_init\_\_(self, name, age, grade, graduation\_year):

super().\_\_init\_\_(name, age, grade)

self.graduation\_year = graduation\_year

def print\_info(self):

print(f"Name: {self.name}, Age: {self.age}, Grade: {self.grade}, Graduation Year: {self.graduation\_year}")

def register\_student(student):

student.print\_info()

# Creating Student and CSEStudent objects

student = Student("Alice", 20, "A")

cse\_student = CSEStudent("David", 23, "A", 2023)

# Registering both students

register\_student(student)

register\_student(cse\_student)