**Write a program for error detecting code using CRC CCITT (16-bits).**

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#define GEN\_POLY 0x1021 // CRC-CCITT polynomial

#define GEN\_POLY\_BIT\_SIZE 16

// Function to calculate CRC-CCITT checksum

uint16\_t calculate\_crc(const char \*message, int message\_length) {

uint16\_t crc = 0xFFFF; // Initialize CRC to 0xFFFF

for (int i = 0; i < message\_length; i++) {

crc ^= (message[i] << (GEN\_POLY\_BIT\_SIZE - 8)); // XOR with the next byte

for (int j = 0; j < 8; j++) {

if (crc & 0x8000) {

crc = (crc << 1) ^ GEN\_POLY;

} else {

crc = crc << 1;

}

}

}

return crc;

}

int main() {

char message[128];

printf("Enter data: ");

scanf(" %[^'\n']s", message); // Read the entire input line including spaces

int message\_length = strlen(message);

char modified\_message[128];

strcpy(modified\_message, message);

// Calculate CRC-CCITT checksum

uint16\_t checksum = calculate\_crc(modified\_message, message\_length);

printf("\nOriginal Message: %s\n", message);

printf("CRC-CCITT Checksum: 0x%04X\n", checksum);

// Test error detection

int error\_choice;

printf("Test error detection (0 for yes, 1 for no): ");

scanf("%d", &error\_choice);

if (error\_choice == 0) {

int error\_position;

printf("Enter the position where error is to be inserted (1-%d): ", message\_length);

scanf("%d", &error\_position);

if (error\_position >= 1 && error\_position <= message\_length) {

modified\_message[error\_position - 1] = (modified\_message[error\_position - 1] == '0') ? '1' : '0';

printf("Modified Message: %s\n", modified\_message);

}

}

// Recalculate CRC-CCITT checksum

uint16\_t new\_checksum = calculate\_crc(modified\_message, message\_length);

if (new\_checksum == 0) {

printf("No error detected.\n");

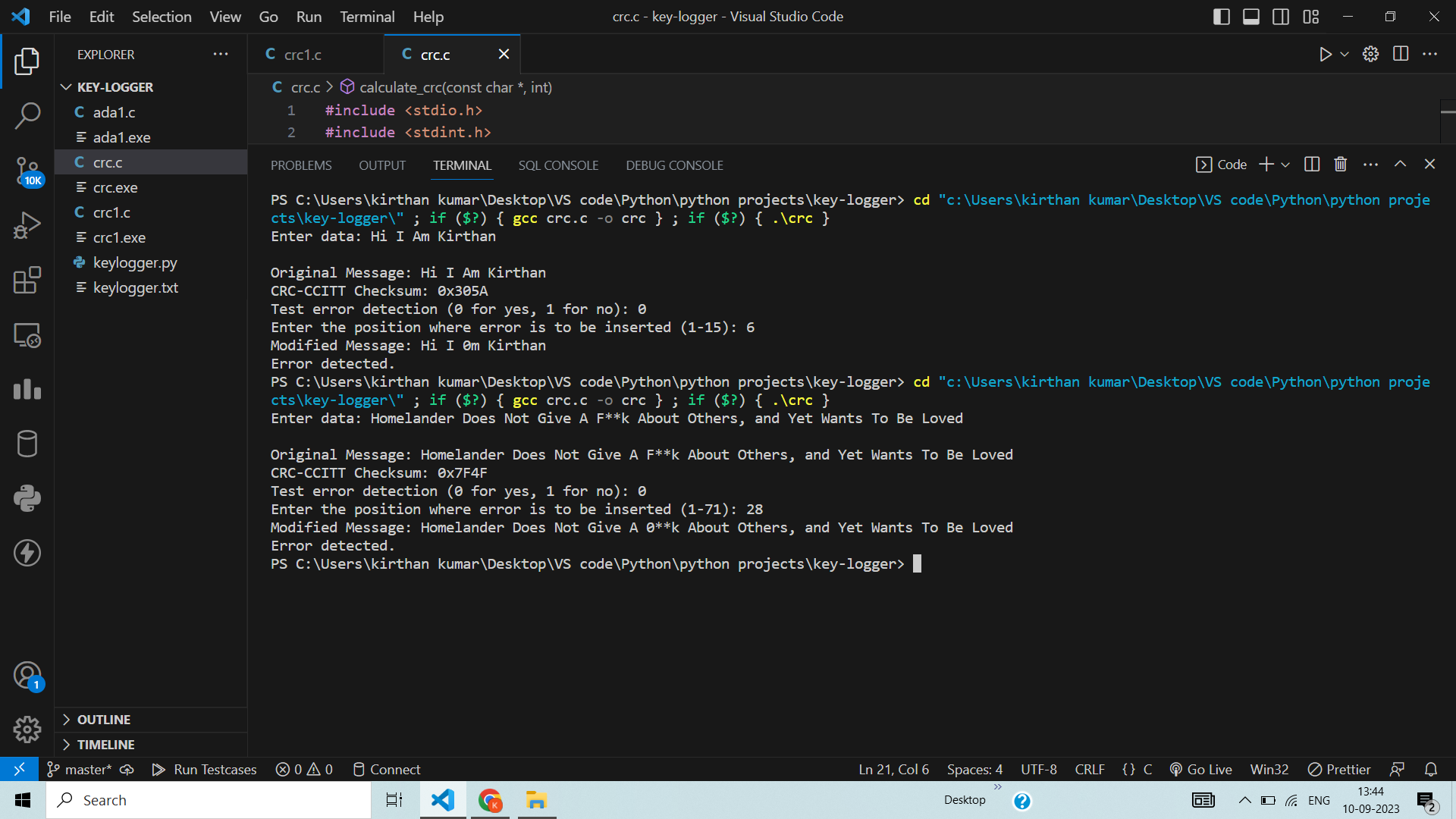
} else {

printf("Error detected.\n");

}

return 0;

}



**Write a program for congestion control using Leaky bucket algorithm.**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <time.h>**

**#define BUCKET\_CAPACITY 10 // Bucket capacity in tokens**

**#define OUTPUT\_RATE 2 // Output rate in tokens per second**

**// Function to perform traffic shaping using the Leaky Bucket algorithm**

**void leaky\_bucket(int input\_rate) {**

**int bucket = 0; // Token bucket initially empty**

**int time = 0; // Simulation time**

**while (1) {**

**printf("\nTime: %d seconds - Bucket: %d tokens\n", time, bucket);**

**// Generate incoming packets**

**int incoming\_packets = rand() % (input\_rate + 1);**

**// Add incoming tokens to the bucket**

**bucket += incoming\_packets;**

**// Limit the bucket capacity**

**if (bucket > BUCKET\_CAPACITY) {**

**bucket = BUCKET\_CAPACITY;**

**}**

**// Transmit data at the output rate**

**int transmitted = (bucket > OUTPUT\_RATE) ? OUTPUT\_RATE : bucket;**

**bucket -= transmitted;**

**// Increment simulation time**

**time++;**

**// Sleep for 1 second to simulate real-time behavior**

**sleep(1);**

**// Check for termination**

**if (time >= 30) {**

**break;**

**}**

**}**

**}**

**int main() {**

**int input\_rate;**

**printf("Enter input rate (packets per second): ");**

**scanf("%d", &input\_rate);**

**srand(time(NULL)); // Seed the random number generator**

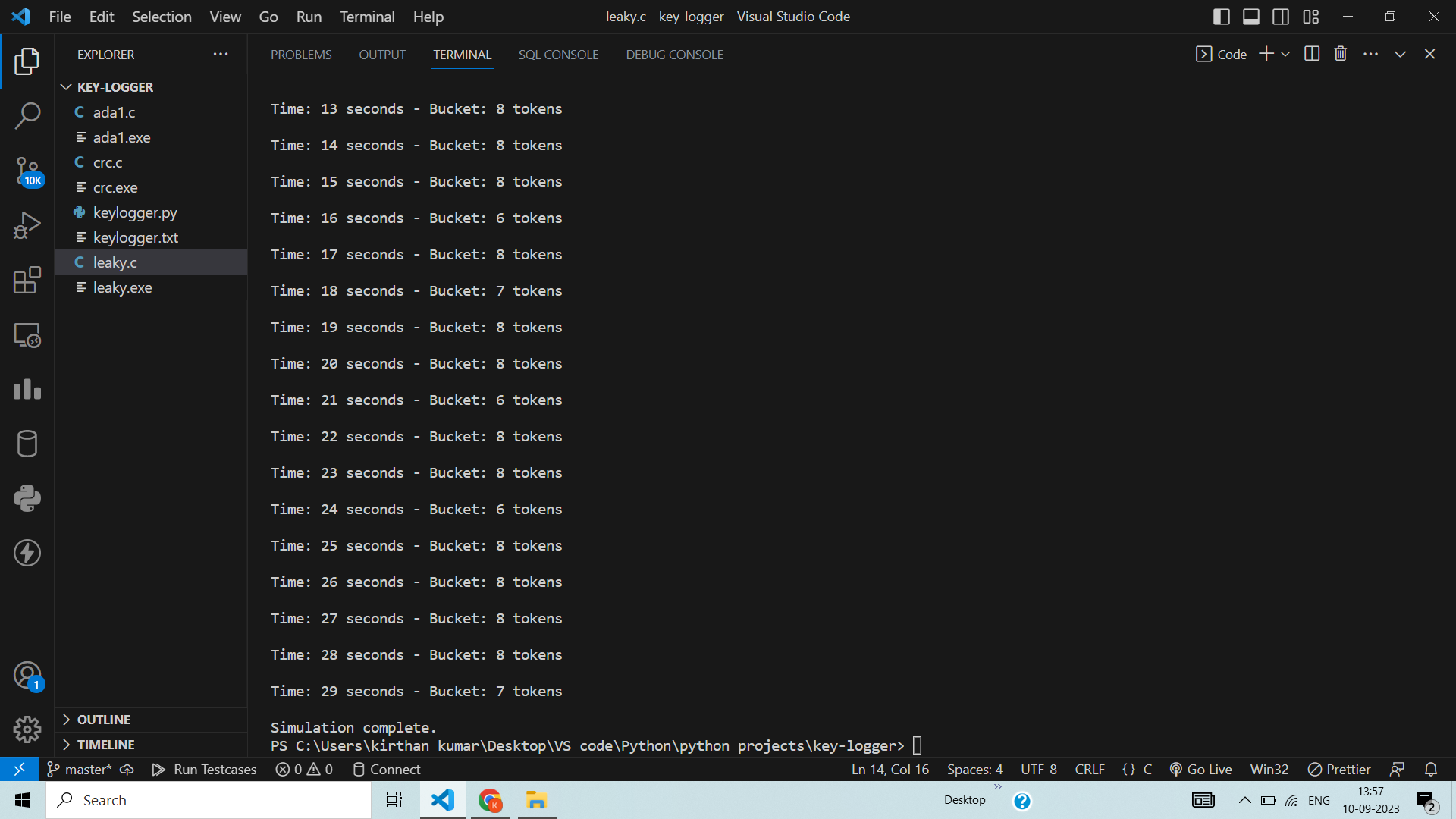
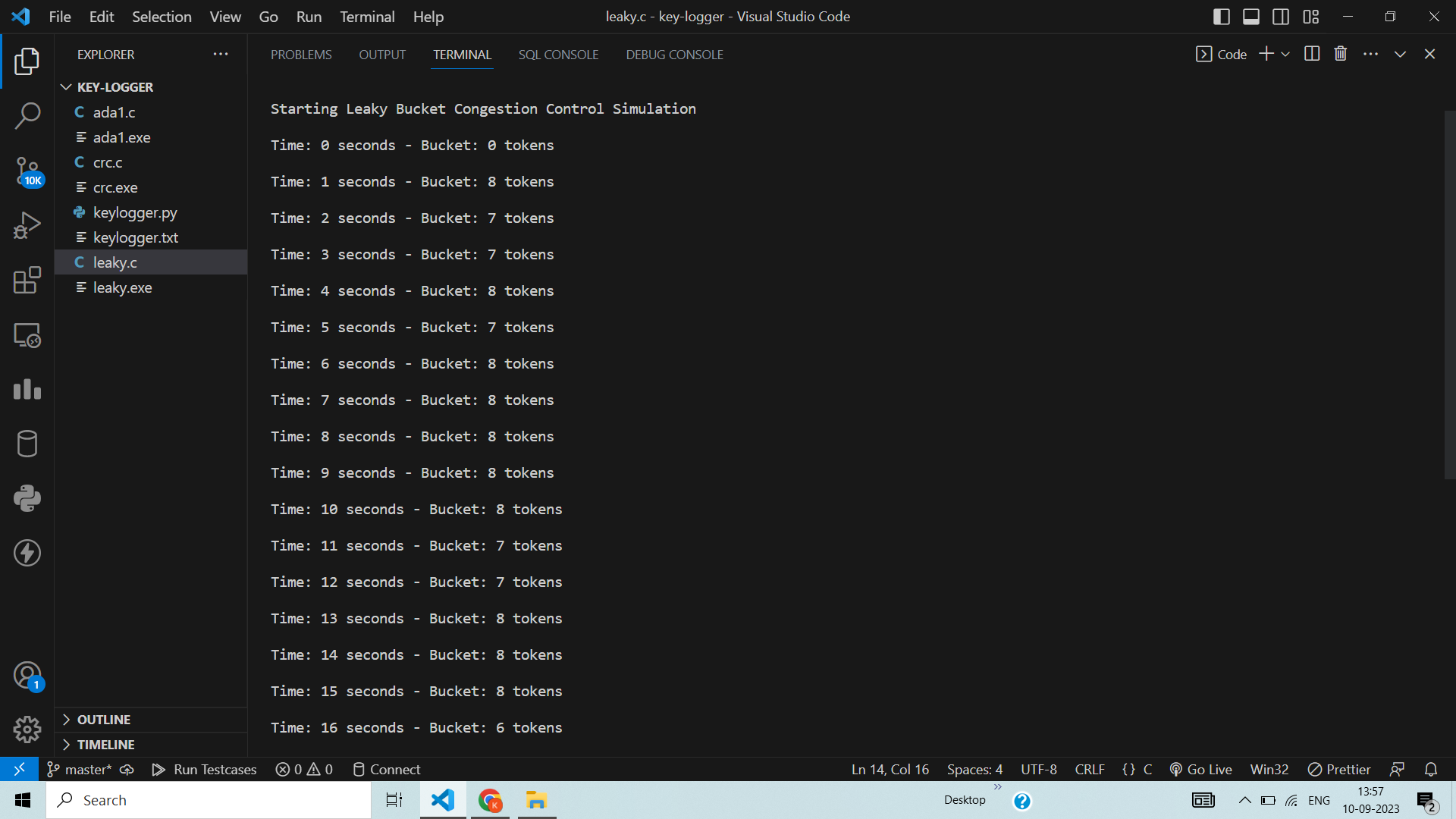
**printf("\nStarting Leaky Bucket Congestion Control Simulation\n");**

**leaky\_bucket(input\_rate);**

**printf("\nSimulation complete.\n");**

**return 0;**

**}**

****

**Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

**serverTCP.py**

**from socket import \***

**serverName = "127.0.0.1"**

**serverPort = 12000**

**serverSocket = socket(AF\_INET, SOCK\_STREAM)**

**serverSocket.bind((serverName, serverPort))**

**serverSocket.listen(1)**

**while 1:**

**print("The server is ready to receive")**

**connectionSocket, addr = serverSocket.accept()**

**sentence = connectionSocket.recv(1024).decode()**

**file = open(sentence, "r")**

**l = file.read(1024)**

**connectionSocket.send(l.encode())**

**print('\nSent contents of ' + sentence)**

**file.close()**

**connectionSocket.close()**

**clientTCP.py**

**from socket import \***

**serverName = '127.0.0.1'**

**serverPort = 12000**

**clientSocket = socket(AF\_INET, SOCK\_STREAM)**

**clientSocket.connect((serverName, serverPort))**

**sentence = input("\nEnter file name: ")**

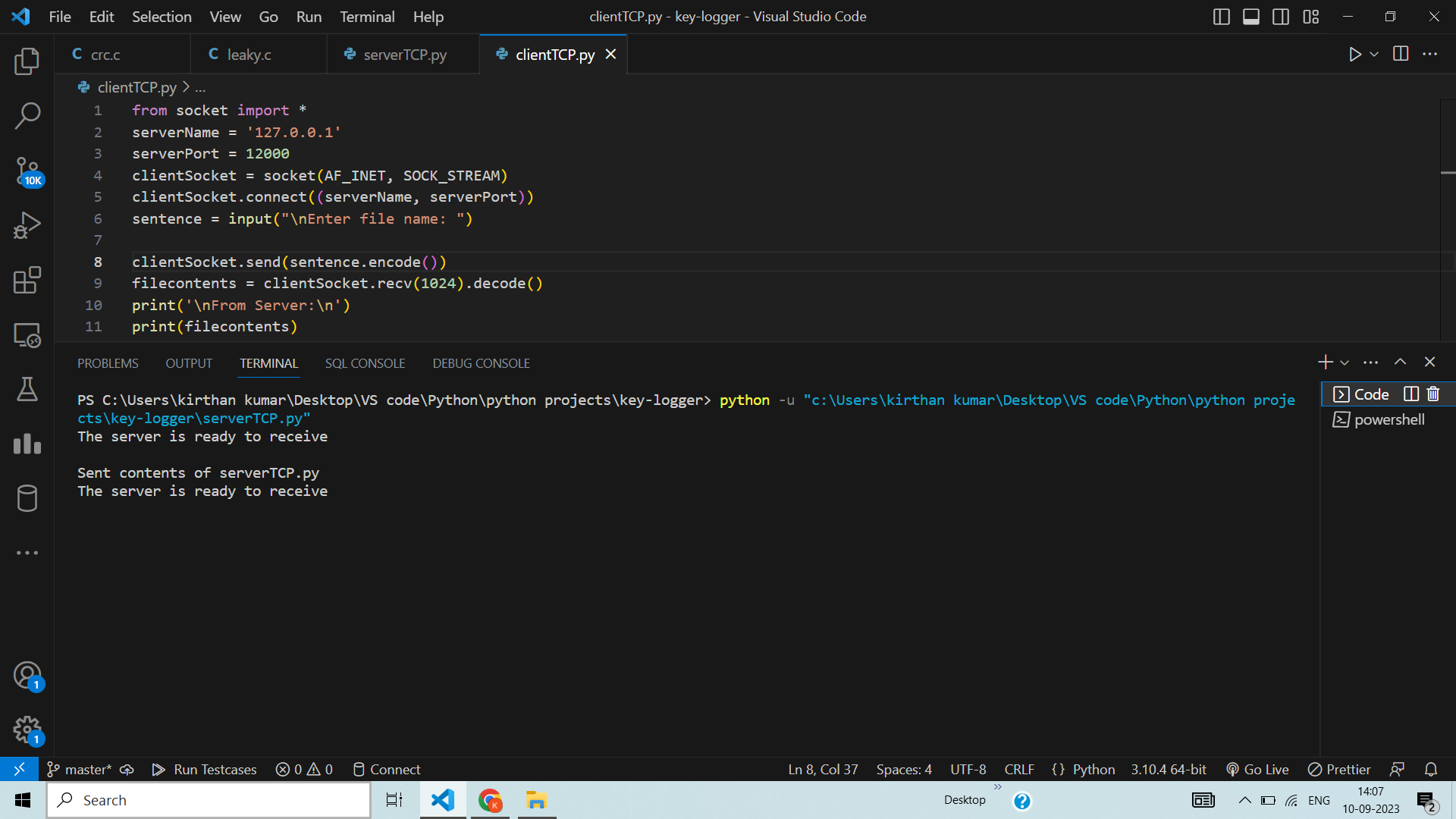
**clientSocket.send(sentence.encode())**

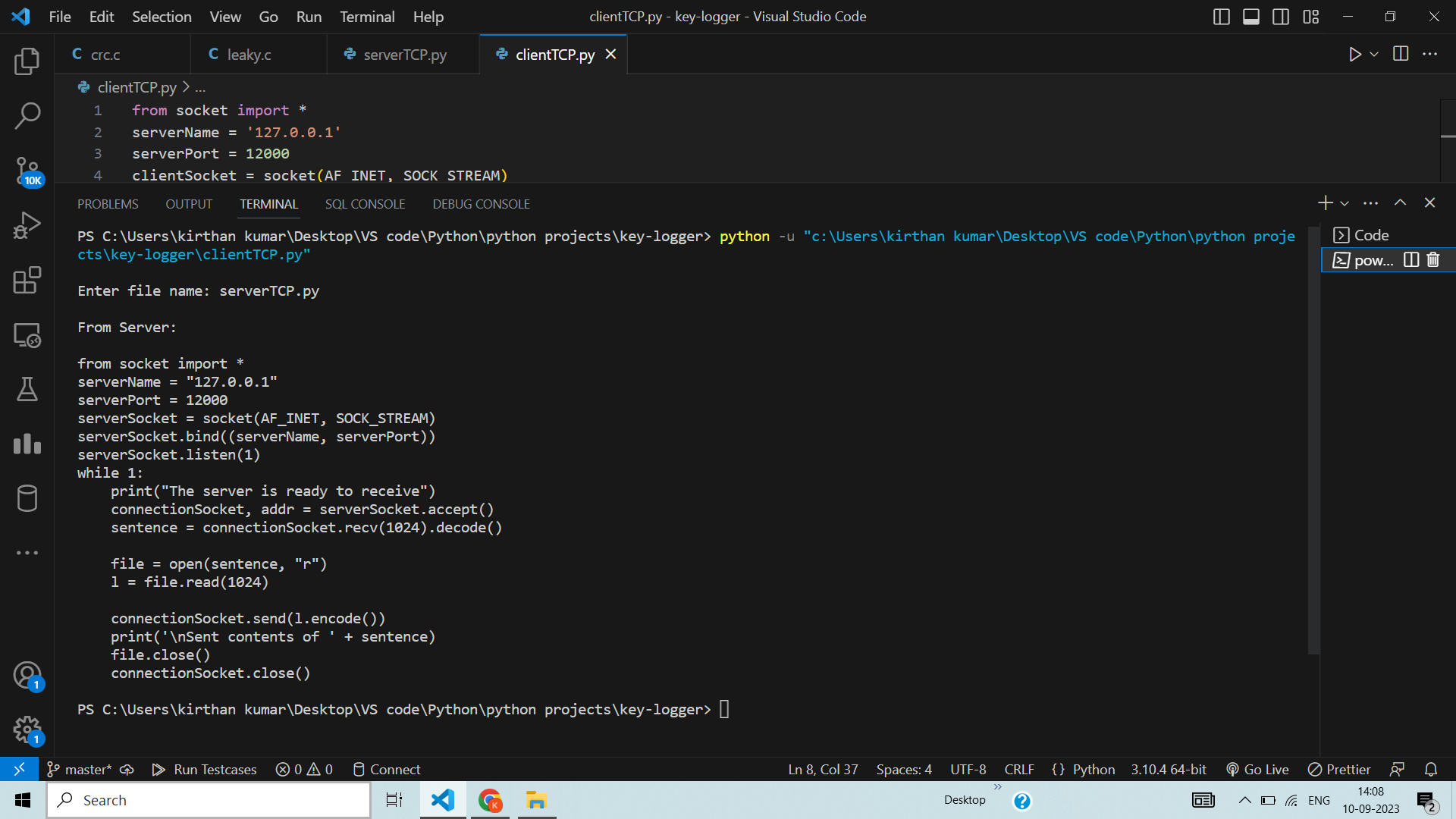
**filecontents = clientSocket.recv(1024).decode()**

**print('\nFrom Server:\n')**

**print(filecontents)**

**clientSocket.close()**

****

****

**Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

**serverUDP.py**

**from socket import \***

**serverPort = 12000**

**serverSocket = socket(AF\_INET, SOCK\_DGRAM)**

**serverSocket.bind(("127.0.0.1", serverPort))**

**print("The server is ready to receive")**

**while 1:**

**sentence, clientAddress = serverSocket.recvfrom(2048)**

**sentence = sentence.decode("utf-8")**

**file = open(sentence, "r")**

**con = file.read(2048)**

**serverSocket.sendto(bytes(con, "utf-8"), clientAddress)**

**print('\nSent contents of ', end=' ')**

**print(sentence)**

**# for i in sentence:**

**# print (str(i), end = '')**

**file.close()**

**clientUDP.py**

**from socket import \***

**serverName = "127.0.0.1"**

**serverPort = 12000**

**clientSocket = socket(AF\_INET, SOCK\_DGRAM)**

**sentence = input("\nEnter file name: ")**

**clientSocket.sendto(bytes(sentence, "utf-8"), (serverName, serverPort))**

**filecontents, serverAddress = clientSocket.recvfrom(2048)**

**print('\nReply from Server:\n')**

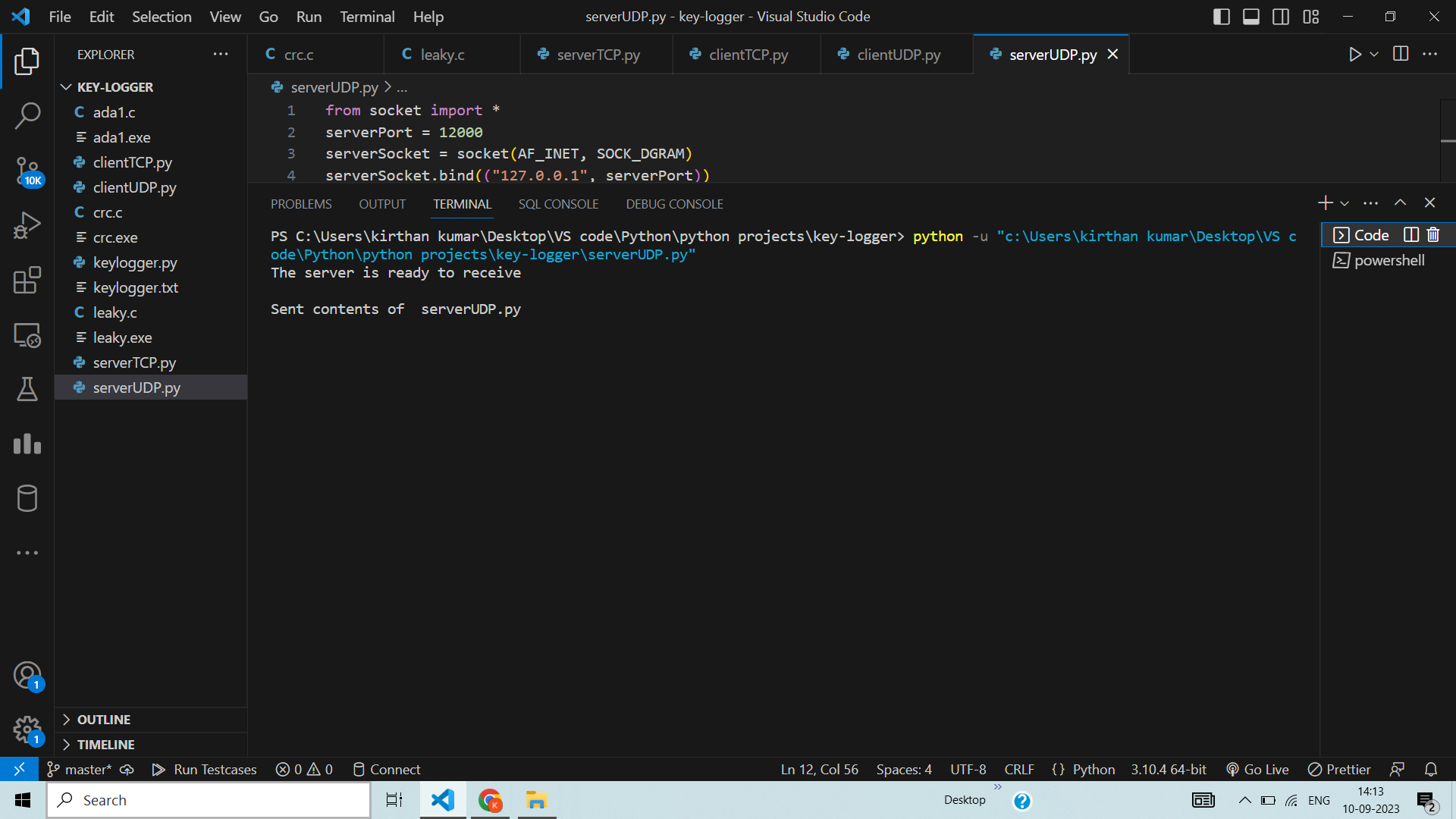
**print(filecontents.decode("utf-8"))**

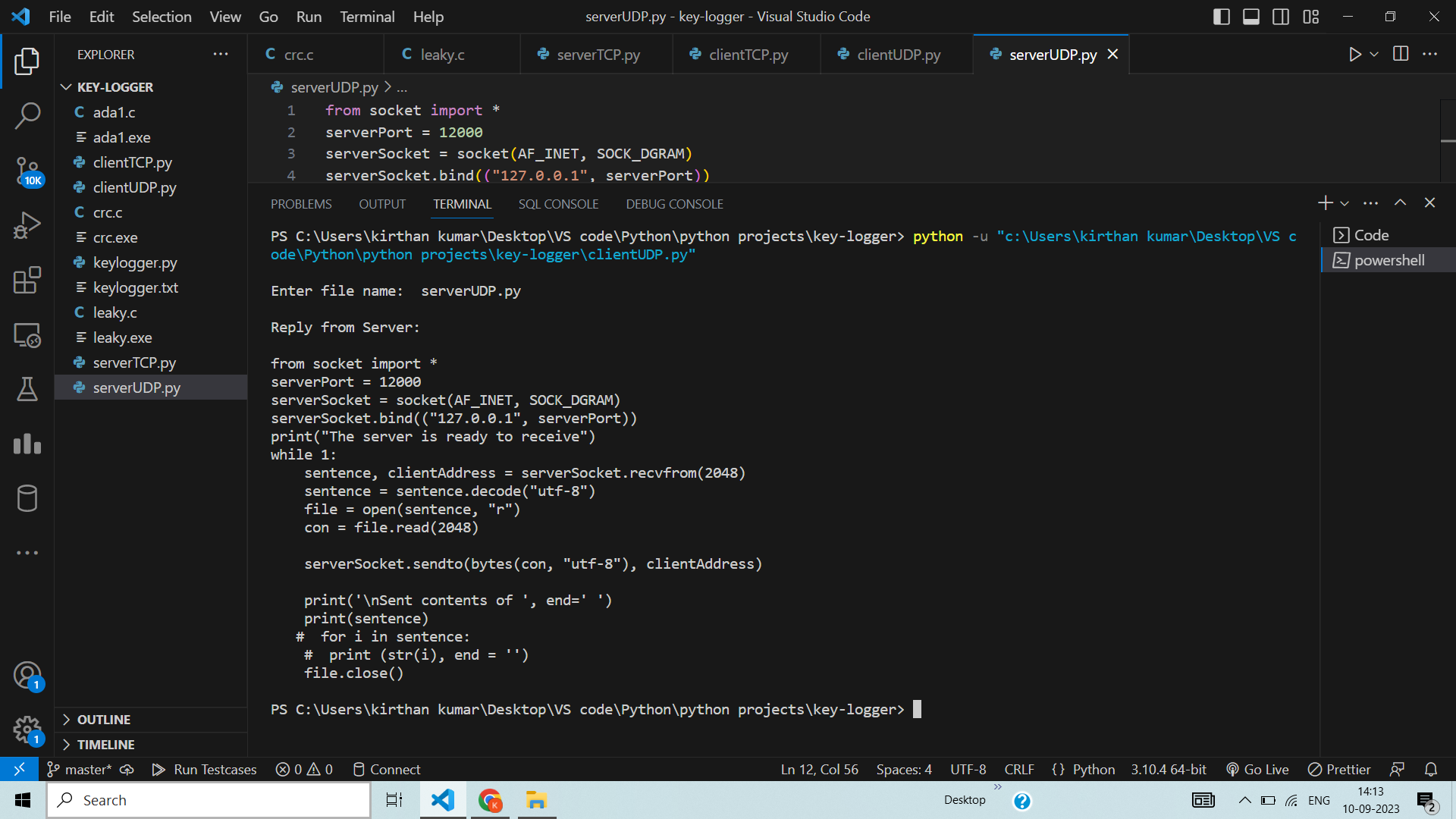
**# for i in filecontents:**

**# print(str(i), end = '')**

**clientSocket.close()**

**clientSocket.close()**

****

****