**Question 1:**

def get\_checksum(word: str):

    # Call the OrdSum Function

    ord\_sum = calculate\_ordsum(word)

    # Call the Checksum Function

    checksum = calculate\_checksum(ord\_sum)

    # Print the checksum value for the input word

    print(f"Checksum for '{word}' is {checksum}")

    # Return the checksum value

    return checksum

def calculate\_checksum(ord\_sum: int):

    # Calculate the checksum by taking modulo 10 of ord\_sum

    checksum = ord\_sum % 10

    # Return the calculated checksum

    return checksum

def calculate\_ordsum(word: str):

    # Initialize variable ord\_sum to 0

    ord\_sum = 0

    # Iterate through each character in word

    for char in word:

        # Add ord value of the character to ord\_sum

        ord\_sum += ord(char)

    # Return the calculated ord\_sum

    return ord\_sum

def main():

    # Input word

    word = "cat"

    # Get the checksum value for the input word

    checksum\_val = get\_checksum(word)

if \_\_name\_\_ == '\_\_main\_\_':

    # Call the main function

    main()

Output:

Case 1:



Case 2:



**Question 2:**

def dispense\_change(change: int):

    # This function takes in the amount of change that is due and calls the calculate\_coin\_count function

    one\_penny, two\_pence, five\_pence, ten\_pence, twenty\_pence, fifty\_pence = calculate\_coin\_count(change)

    return one\_penny, two\_pence, five\_pence, ten\_pence, twenty\_pence, fifty\_pence

def calculate\_coin\_count(change: int):

    # This function calls all the specific coin functions to calculate the number of each coin denomination and returns them and the remaining change

    remaining\_change = change

    fifty\_pence, remaining\_change = calculate\_fifty\_pence(remaining\_change)

    twenty\_pence, remaining\_change = calculate\_twenty\_pence(remaining\_change)

    ten\_pence, remaining\_change = calculate\_ten\_pence(remaining\_change)

    five\_pence, remaining\_change = calculate\_five\_pence(remaining\_change)

    two\_pence, remaining\_change = calculate\_two\_pence(remaining\_change)

    one\_penny, remaining\_change = calculate\_one\_penny(remaining\_change)

    return one\_penny, two\_pence, five\_pence, ten\_pence, twenty\_pence, fifty\_pence

def calculate\_one\_penny(change: int):

    '''

    calculate\_one\_penny(1)

    returns 1,0

    calculate\_one\_penny(0)

    returns 0,0

    '''

    # This function calculates the number of one penny coins required for the given change

    number\_of\_coins = change % 2

    new\_change\_value = (change - number\_of\_coins\*1)

    return number\_of\_coins, new\_change\_value

def calculate\_two\_pence(change: int):

    '''

    calculate\_two\_pence(2)

    returns 1,0

    calculate\_two\_pence(1)

    returns 0,1

    '''

    # This function calculates the number of two pence coins required for the given change

    number\_of\_coins = change // 2

    new\_change\_value = (change - number\_of\_coins\*2)

    return number\_of\_coins, new\_change\_value

def calculate\_five\_pence(change: int):

    '''

    calculate\_five\_pence(3)

    returns 0,3

    calculate\_five\_pence(8)

    returns 1,3

    '''

    # This function calculates the number of five pence coins required for the given change

    number\_of\_coins = change // 5

    new\_change\_value = (change - number\_of\_coins\*5)

    return number\_of\_coins, new\_change\_value

def calculate\_ten\_pence(change: int):

    '''

    calculate\_ten\_pence(3)

    returns 0,3

    calculate\_ten\_pence(27)

    returns 2,7

    '''

    # This function calculates the number of ten pence coins required for the given change

    number\_of\_coins = change // 10

    new\_change\_value = (change - number\_of\_coins\*10)

    return number\_of\_coins, new\_change\_value

def calculate\_twenty\_pence(change: int):

    '''

    calculate\_twenty\_pence(3)

    returns 0,3

    calculate\_twenty\_pence(27)

    returns 1,7

    '''

    # This function calculates the number of twenty pence coins required for the given change

    number\_of\_coins = change // 20

    new\_change\_value = (change - number\_of\_coins\*20)

    return number\_of\_coins, new\_change\_value

def calculate\_fifty\_pence(change: int):

    '''

    calculate\_fifty\_pence(2)

    returns 0,2

    calculate\_fifty\_pence(63)

    returns 1,13

    '''

    # This function calculates the number of fifty pence coins required for the given change

    number\_of\_coins = change // 50

    new\_change\_value = (change - number\_of\_coins\*50)

    return number\_of\_coins, new\_change\_value

def main():

    change = 99

    if change > 99:

        print("Change must be below 99")

    else:

        one\_penny, two\_pence, five\_pence, ten\_pence, twenty\_pence, fifty\_pence = dispense\_change(change)

        print(f"Change: {change}p")

        print(f"1p: {one\_penny}")

        print(f"2p: {two\_pence}")

        print(f"5p: {five\_pence}")

        print(f"10p: {ten\_pence}")

        print(f"20p: {twenty\_pence}")

        print(f"50p: {fifty\_pence}")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Output:

Case 1:

Text

Description automatically generated

Case 2:

Text

Description automatically generated

Question 3:

import os

import sys

import csv

import pprint

def clean\_data(raw\_data):

    raw\_data = raw\_data.replace("\t", ",")

    raw\_data = raw\_data.split("\n")

    for counter, row in enumerate(raw\_data):

        if counter == 0:

            y = [x.strip() for x in row.split(",")]

            deats = y[1].split("(")

            y[1] = deats[0]

            y.insert(2, deats[1].split("/")[0])

            y[4] = y[4][4:]

            y[7] = y[7][:-1]

            raw\_data[counter] = ",".join(y)

        else:

            columns = [x.strip() for x in row.split(",")]

            new\_part = columns[0].split(" ")

            if len(new\_part) == 3:

                new\_part[0], new\_part[1] = new\_part[1], new\_part[0]

            if len(new\_part) == 4:

                new\_part[0], new\_part[1] = f"{new\_part[1]} {new\_part[2]}", new\_part[0]

                new\_part.pop(2)

            new\_part[2] = new\_part[2].replace("(", "").replace(")", "")

            columns.pop(0)

            raw\_data[counter] = ",".join([\*new\_part, \*columns])

    cleaned\_data = list(csv.DictReader(raw\_data))

    for row in cleaned\_data:

        row['Last Name'] = row['Last Name'].upper()

        row['First Name'] = row['First Name'].upper()

    return cleaned\_data

def read\_file():

    # file\_name = input("Enter File Name (Make sure it's in the same directory): ")

    file\_name = "marathon\_London.txt"

    try:

        with open((os.path.join(sys.path[0], file\_name)), 'r') as input\_file:

            raw\_data = input\_file.read()

        return clean\_data(raw\_data)

    except FileNotFoundError: # Errors could be raised if the user does not type the right file name these should be handled

        raise FileNotFoundError("Please Enter a Valid File Name")

def lineCount(data):

    number\_of\_lines = len(data) # Since we have put the data cleanly into a dictionary we can simply call len

    print(f"Number of Lines in Input File is {number\_of\_lines}") # Print the len

def runnerCountry(data):

    runner\_name = input("Enter the runner's name (In Capitals): ") # Getting User input

    data = {x["Last Name"]:x for x in data}

    try:

        print(f"This runner is from {data[runner\_name]['Country']}")

    except KeyError:

        raise Exception("No Valid Runner Name entered")

def countryInformation(data):

    all\_countries = list(set([row["Country"] for row in data])) # Giving the user the options

    print(f"Printing all countries in list: {all\_countries}")

    country = input("Enter the country you want information from: ") # Getting User input

    if country not in all\_countries:

        raise Exception("You must choose a country in the list")

    output = [row for row in data if row["Country"] == country] # Finding all the rows the user asked for

    print(f"{len(output)} Rows matching country user specified")

    for row in output:

        pprint.pprint(row)

def format\_hhmmss\_to\_seconds(time\_str):

    """Get seconds from time."""

    h, m, s = time\_str.split(':')

    return int(h) \* 3600 + int(m) \* 60 + int(s)

def format\_seconds\_to\_hhmmss(seconds):

    hours = seconds // (60\*60)

    seconds %= (60\*60)

    minutes = seconds // 60

    seconds %= 60

    return f"{hours:02.0f}:{minutes:02.0f}:{seconds:02.0f}"

def averageTime(data):

    all\_times = []

    for row in data:

        time = format\_hhmmss\_to\_seconds(row['Net Time'])

        all\_times.append(time)

    average\_time\_secs = sum(all\_times)/len(data)

    average\_time\_formatted = format\_seconds\_to\_hhmmss(average\_time\_secs)

    print(f"The average time across all runners is {average\_time\_secs} in seconds. {average\_time\_formatted} in the format provided in the data")

def main():

    data = read\_file()

    lineCount(data)

    runnerCountry(data)

    countryInformation(data)

    averageTime(data)

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

    main()

Output:

Case 1:

Text

Description automatically generated

Case 2:

Text

Description automatically generated

Case 3 (if user enters bad input):

A screenshot of a computer

Description automatically generated with medium confidence

Raises the correct error.

Question 4:

"""

The program uses a Monte Carlo simulation to estimate the probability that more than 20% of the farmer's crop

will be affected by pests. It does this by simulating the sampling process multiple times (10,000 simulations) and for

each simulation, it randomly selects a sample of 1000 plants from the entire crop, calculates the proportion

of affected plants in that sample, and records whether the proportion of affected plants in that sample is

greater than 20%. After all the simulations are completed, it calculates the percentage of times that the proportion

of affected plants was greater than 20% and compares it to the actual proportion derived from the sample data.

"""

import random

def simulate\_pest\_effect(predicted\_percentage\_affected: float, number\_of\_plants\_taken\_on\_each\_simulation: int, total\_number\_of\_plants: int, num\_simulations: int, cutoff: float):

    """

    The function simulates the pest proportion of the plants by taking random samples of plants on each simulation.

    The function checks if the proportion of affected plants in the sample is greater than the cutoff (20%)

    The function returns the probability of pest proportion greater than the cutoff(20%)

    """

    # Initialize variables to store the number of simulations where pest proportion is greater than 20%

    # and the number of affected plants in each simulation

    greater\_than\_cutoff = 0

    affected\_plants = 0

    for simulation\_number in range(num\_simulations):

        if simulation\_number % 1000 == 0:

            print(f"Finished {simulation\_number} Simulations")

        # Use random number generator to randomly select plants for the simulation

        plants\_sample = random.sample(range(total\_number\_of\_plants), number\_of\_plants\_taken\_on\_each\_simulation)

        affected\_plants = sum([1 for i in plants\_sample if i < predicted\_percentage\_affected\*total\_number\_of\_plants])

        # Check if proportion of affected plants in the sample is greater than the cutoff (20%)

        if affected\_plants/number\_of\_plants\_taken\_on\_each\_simulation > cutoff:

            greater\_than\_cutoff += 1

    # Calculate the probability of pest proportion greater than the 20% by seeing the number of times the simulation ran above the cutoff

    probability = float(greater\_than\_cutoff)/float(num\_simulations)

    return f"{probability:.2%}"

def main():

    """

    The main function runs the simulation, call the simulate\_pest\_effect function and prints the result

    """

    cutoff = 0.2 # %20 from the question

    predicted\_percentage\_affected = 0.22 # 22% infected from the farmer's first model

    number\_of\_plants\_taken\_on\_each\_simulation = 1000 #number of plants on each run

    total\_number\_of\_plants = 10000 # number of plants in field we're just modelling a small portion

    number\_of\_simulations\_to\_run = 10000 # number of simulations to run

    probability = simulate\_pest\_effect(predicted\_percentage\_affected, number\_of\_plants\_taken\_on\_each\_simulation, total\_number\_of\_plants, number\_of\_simulations\_to\_run, cutoff)

    print(f"Probability of pest proportion greater than 20%: {probability}")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Output:

Case 1:

Graphical user interface, text

Description automatically generated

Case 2:

Text

Description automatically generated