# CA-2

Anmol Gupta

2107052013

Q1. Generate a model for Covid 19 with symptoms of parameters like fever, cold, shivering, weight loss, generate 100 model data with random values for each parameter and order by parameter lowest to highest in display based on the input parameter.

```
import random


class CovidPatient:
    def __init__(self, patient_id, fever, cold, shivering, weight_loss):
        self.patient_id = patient_id
        self.fever = fever
        self.cold = cold
        self.shivering = shivering
        self.weight_loss = weight_loss


    def __repr__(self):
        return (f"Patient {self.patient_id} - Fever: {self.fever}, Cold: {self.cold}, "
                f"Shivering: {self.shivering}, Weight Loss: {self.weight_loss}")


def generate_patients(num_patients, seed_value):
    random.seed(seed_value)
```

```
    patients = []
    for i in range(1, num_patients + 1):
        fever = random.uniform(98.0, 105.0)
        cold = random.randint(0, 10)
        shivering = random.randint(0, 10)
        weight_loss = random.uniform(0, 20)
        patient = CovidPatient(i, fever, cold, shivering, weight_loss)
        patients.append(patient)
    return patients

def order_patients_by_parameter(patients, parameter):
    if parameter == 'fever':
        patients.sort(key=lambda p: p.fever)
    elif parameter == 'cold':
        patients.sort(key=lambda p: p.cold)
    elif parameter == 'shivering':
        patients.sort(key=lambda p: p.shivering)
    elif parameter == 'weight_loss':
        patients.sort(key=lambda p: p.weight_loss)
    return patients

NUM_PATIENTS = 100
SEED_VALUE = 42

patients = generate_patients(NUM_PATIENTS, SEED_VALUE)
```

```
parameter = input("Enter the parameter to sort by (fever, cold, shivering, weight_loss): ").lower()

ordered_patients = order_patients_by_parameter(patients, parameter)

for patient in ordered_patients:
    print(patient)
```

Q2. Generate a model to represent a mathematical equation, write a program to parse the equation, and ask for input for each parameter

```
import random

class SavingsAccount:
    def __init__(self, account_number, initial_balance=0):
        self.account_number = account_number
        self.balance = initial_balance
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(f"Deposit: +{amount}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transactions.append(f"Withdraw: -{amount}")
```

```
        else:
            self.transactions.append(f"Withdraw failed (Insufficient funds): -{amount}")

    def __repr__(self):
        return f"Account {self.account_number} - Balance: {self.balance}"

def generate_random_transactions(account, num_months, num_transactions_per_month, seed_value):
    random.seed(seed_value)
    for _ in range(num_months):
        for _ in range(num_transactions_per_month):
            transaction_type = random.choice(['deposit', 'withdraw'])
            amount = random.randint(1, 1000)

            if transaction_type == 'deposit':
                account.deposit(amount)
            else:
                account.withdraw(amount)

def generate_accounts(num_accounts, num_months, num_transactions, seed_value):
    accounts = []
    for i in range(1, num_accounts + 1):
        initial_balance = random.randint(1000, 10000)
        account = SavingsAccount(account_number=i, initial_balance=initial_balance)
```

```
        generate_random_transactions(account, num_months, num_transactions,
seed_value)
        accounts.append(account)

    accounts.sort(key=lambda acc: acc.balance)

    return accounts

NUM_ACCOUNTS = 100
NUM_MONTHS = 12
NUM_TRANSACTIONS_PER_MONTH = 10
SEED_VALUE = 42

accounts = generate_accounts(NUM_ACCOUNTS, NUM_MONTHS,
NUM_TRANSACTIONS_PER_MONTH, SEED_VALUE)

for account in accounts:
    print(account)
```