**Pseudocode and diagrams**

**Task 1**

**Import packages**

IMPORT CSV

IMPORT datetime

From datetime import date

**Class**

Class Patient:

FUNCTION INITIALISE Attributes of patient object:

(self, patient\_name, date\_of\_birth, sex, height\_of\_patient, weight\_of\_patient,build)

self.calculateBMI = None  
self.Patient\_name = patient\_name self.date\_of\_birth = date\_of\_birth self.Patient\_sex = sex

self.build\_of\_patient = build

self.height\_of\_patient = float(height\_of\_patient) self.weight\_of\_patient = int(weight\_of\_patient) self.patients\_BMI = self.weight\_of\_patient / (self.height\_of\_patient \*\* 2) self.patients\_age = calculate\_patients\_age(date\_of\_birth)

GET patients name

RETURN Patients name

GET patients age

RETURN patients age

GET patients BMI

Return patients BMI

GET weight class

RETURN class of patients weight

class\_of\_patients\_weight = “”

**Helper functions**

calculate\_patients\_age(date\_of\_birth)

CREATE Date\_of\_birth\_list = date\_of\_birth.split(“/”)

CREATE an integer value to separate the year of the patient

CREATE an integer value to separate the month of the patient

CREATE an integer value to separate the day of the patient

CREATE a date variable to be stored of the patient’s date of birth –

date\_of\_birth = datetime.date(day, month, year)

ADD date of today

RETURN ((date of today – date of birth) /365).days

set\_class\_weight\_of\_patient(patient):

IF patients BMI is less than 18.5

SET class of patients weight = “Underweight”

ELIF if patients BMI is less than or equal to 18.5 or less than or equal to 25

SET class patients weight = “Normal”

IF patients less than 25 or less than or equal to 28

SET class patients weight = “Overweight”

ELIF patients weight more than 28

SET class patients weight = “Obese”

readCSV():

CREATE empty list called List\_of\_patient

OPEN the csv called **"DADSA 2021 CWK B DATA COLLECTION.csv"**, and set it as patientData

LOOP through each for row in csvreader:

CREATE patient = Patient(row[0], row[1], row[2], row[3], row[4],row[5]) –  *A new patient object is discovered and all data will be set by its constructor*

APPEND the patient to the list of patients

CLOSE patientData

RETURN List\_of\_patient

**Print Statements**

Patient\_fields\_task1(patient)

PRINT (Patients name, patients age, patients build, patients BMI, patients weight class)

order\_of\_weight\_patients(obese\_patients, underweight\_patients, overweight\_patients, normalweight\_patients):

Weight\_counter = 0

PRINT ("Here is a list of patients and are classed by their significant weight")  
PRINT ("Every 10 patients are listed for you to examine, please look through this carefully!")  
INPUT("Please press enter to proceed for task 1. \n")

LOOP through every patient in obese\_patients

CALL patients\_fields\_task1

Increment by 1 if patient is obese

IF Weight\_counter has more than 10 patients

PRINT new line

LOOP through every patient in underweight\_patients

CALL patients\_fields\_task1

Increment by 1 if patient is underweight

IF Weight\_counter has more than 10 patients

PRINT new line

LOOP through every patient in overweight\_patients

CALL patients\_fields\_task1

Increment by 1 if patient is overweight

IF Weight\_counter has more than 10 patients

PRINT new line

LOOP through every patient in normal\_patients

CALL patients\_fields\_task1

Increment by 1 if patient is normal

IF Weight\_counter has more than 10 patients

PRINT new line

list\_five\_worst\_patients(underweight\_patients, obese\_patients):

CREATE empty list called under\_weight\_male\_patients

CREATE empty list called under\_weight\_female\_patients

CREATE empty list called obese\_male\_patients

CREATE empty list called obese\_female\_patients

LOOP through patient in underweight\_patients

IF it has “M” through the patient.patient\_sex class

SET it as under\_weight\_male\_patients

ELSE under\_weight\_female\_patients

LOOP through patient in obese\_patients

IF it has “M” through the patient.patient\_sex class

SET it as obese\_male\_patients

ELSE obese\_female\_patients

*Sorts out the subgroups by their BMI using lambda expression and the reverse method -*

under\_weight\_male\_patients.sort(key=lambda patient: patient.patients\_BMI, reverse=True)   
under\_weight\_female\_patients.sort(key=lambda patient: patient.patients\_BMI, reverse=True)  
obese\_male\_patients.sort(key=lambda patient: patient.patients\_BMI, reverse=True)  
obese\_female\_patients.sort(key=lambda patient: patient.patients\_BMI, reverse=True)

INPUT("Please press enter to display the worse underweight male patients \n")  
  
PRINT("Worst underweight male patients: \n")  
LOOP through weight in under\_weight\_male\_patients:  
CALL patient\_fields\_task1(weight)  
PRINT new line   
  
INPUT("Please press enter to display the worse underweight female patients \n")  
  
PRINT("Worst underweight female patients: \n")  
LOOP through weight in under\_weight\_female\_patients:  
CALL patient\_fields\_task1(weight)  
PRINT new line  
  
INPUT("Please press enter to display the worse obese male patients \n")  
  
PRINT ("Worst obese male patients: \n")  
LOOP through weight in obese\_male\_patients:  
CALL patient\_fields\_task1(weight)  
print new line  
  
INPUT ("Please press enter to display the worse obese female patients \n")  
  
PRINT ("Worst obese female patients: \n")  
LOOP through weight in obese\_female\_patients:  
CALL patient\_fields\_task1(weight)  
print new line

categorising\_weight\_of\_patients():

CREATE empty list called underweight\_patients

CREATE empty list called normaleight\_patients

CREATE empty list called overweight\_patients

CREATE empty list called obese\_patients

CALL read\_csv to create a list data structure containing objects of the patient class called patient\_list

LOOP through patient in patient\_list

SET class\_weight\_of\_patient

IF "Underweight" == patient.class\_of\_patients\_weight:  
APPEND underweight\_patients list  
ELIF "Normal" == patient.class\_of\_patients\_weight:  
APPEND normal weight\_patients list  
ELIF"Overweight" == patient.class\_of\_patients\_weight:  
APPEND overweight\_patients list   
ELIF"Obese" == patient.class\_of\_patients\_weight:  
APPEND obese\_patients list

CALL order\_of\_weight\_patients(obese\_patients, underweight\_patients, overweight\_patients, normalweight\_patients)  
CALL list\_five\_worst\_patients(underweight\_patients, obese\_patients)

CALL categorising\_weight\_of\_patients()

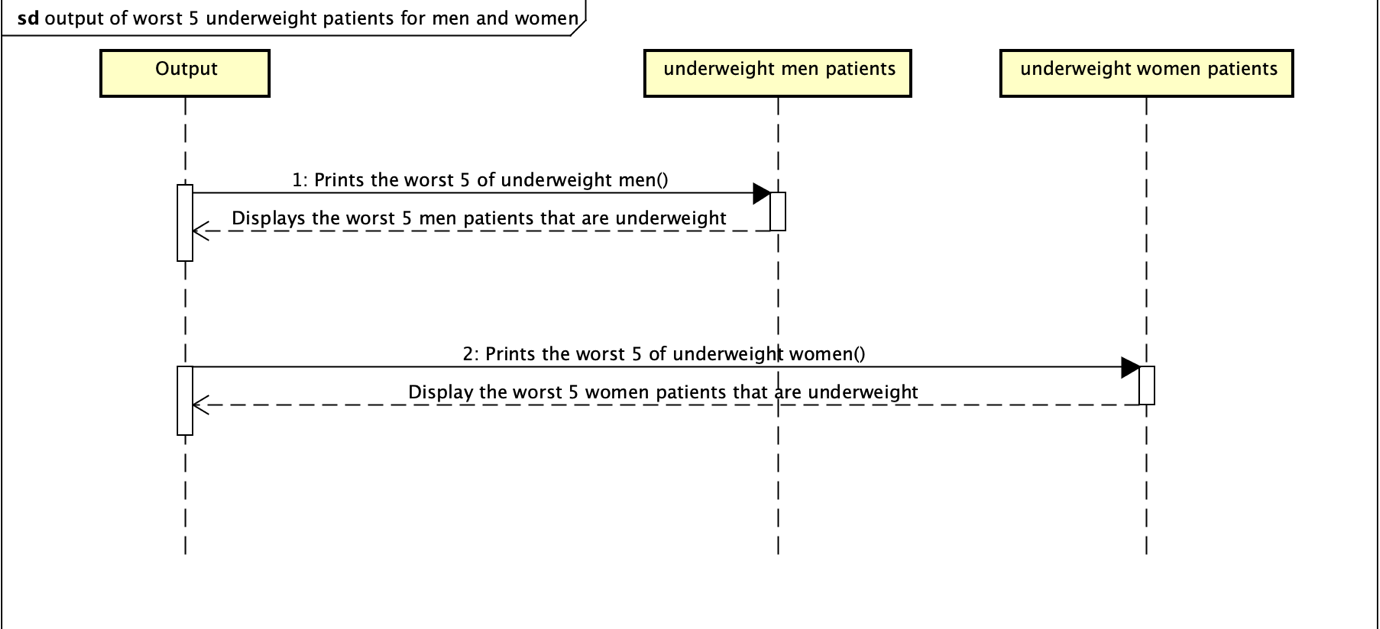
**Diagrams for task 1**

Diagram

Description automatically generated

Diagram

Description automatically generated with medium confidence



**Task 2**

**Import packages**

IMPORT CSV

IMPORT datetime

From datetime import date

PRINT("Here is a list of patients and are classed by oldest age to youngest age whether they need to be referred to dietitian")  
PRINT("Every 10 patients are listed for you to examine, please look through this carefully!")  
INPUT("Please press enter to proceed for task 2. \n")

**Class**

Class Patient:

FUNCTION INITIALISE Attributes of patient object:

(self, patient\_name, date\_of\_birth, sex, height\_of\_patient, weight\_of\_patient,build)

self.calculateBMI = None  
self.Patient\_name = patient\_name self.date\_of\_birth = date\_of\_birth self.Patient\_sex = sex

self.build\_of\_patient = build self.height\_of\_patient = float(height\_of\_patient) self.weight\_of\_patient = int(weight\_of\_patient) self.patients\_BMI = self.weight\_of\_patient / (self.height\_of\_patient \*\* 2) self.patients\_age = calculate\_patients\_age(date\_of\_birth)

GET patients name

RETURN Patients name

GET patients age

RETURN patients age

GET patients BMI

Return patients BMI

GET weight class

RETURN class of patients weight

CREATE empty list called smoker

CREATE empty list called Asthmatic

CREATE empty list called NJT\_or\_NGR

CREATE empty list called Hypertension

CREATE empty list called Renal\_RT

CREATE empty list called Ileostomy\_or\_Colostomy

CREATE empty list called Parenteral\_Nutrition

class\_of\_patients\_weight = “”

SET topPriority to False

**Helper functions**

calculate\_patients\_age(date\_of\_birth)

CREATE Date\_of\_birth\_list = date\_of\_birth.split(“/”)

CREATE an integer value to separate the year of the patient

CREATE an integer value to separate the month of the patient

CREATE an integer value to separate the day of the patient

CREATE a date variable to be stored of the patient’s date of birth –

date\_of\_birth = datetime.date(day, month, year)

ADD date of today

RETURN ((date of today – date of birth) /365).days

set\_class\_weight\_of\_patient(patient):

IF patients BMI is less than 18.5

SET class of patients weight = “Underweight”

ELIF if patients BMI is less than or equal to 18.5 or less than or equal to 25

SET class patients weight = “Normal”

IF patients less than 25 or less than or equal to 28

SET class patients weight = “Overweight”

ELIF patients weight more than 28

SET class patients weight = “Obese”

readCSV():

CREATE empty list called List\_of\_patient

OPEN the csv called **"DADSA 2021 CWK B DATA COLLECTION.csv"**, and set it as patientData

LOOP through each for row in csvreader:

CREATE patient = Patient(row[0], row[1], row[2], row[3], row[4],row[5]) –  *A new patient object is discovered and all data will be set by its constructor*

IF row[6] != **' '**:  
SET patient.Smoker as True  
IF row[7] != **''**:  
SET patient.Asthmatic as True  
IF row[8] != **' '**:

SET patient.NJT\_or\_NGR as True  
IF row[9] != **' '**:

SET patient.Hypertension as True  
IF row[10] != **''**:  
SET patient.Renal\_RT as True  
IF row[11] != **''**:  
SET patient.Ileostomy\_or\_Colostomy as True  
IF row[12] != **''**:  
SET patient.Parenteral\_Nutrition as True

APPEND the patient to the list of patients

CLOSE patientData

RETURN List\_of\_patient

**Print Statements**

Patient\_fields\_task2(patient)

PRINT (Patients name, patients age, patients build, patients BMI, patients weight class, Top priority)

refer\_or\_not(referral\_list):

CREATE an empty list called refer\_or\_not\_list

LOOP through patient in referral\_list

IF (  
 *# uses DeMorgan's law to provide specific Boolean statements  
 # which can be written in different ways for the same effect.* not (not (  
 patient.class\_of\_patients\_weight == "Obese"or patient.class\_of\_patients\_weight ==  
 "Underweight")  
 and not (  
 patient.Hypertension  
 or patient.Asthmatic  
 or patient.Smoker  
 or patient.NJT\_or\_NGR  
 or patient.Renal\_RT  
 or patient.Ileostomy\_or\_Colostomy  
 or patient.Parenteral\_Nutrition))):

APPEND refer\_or\_not \_list   
  
RETURN refer\_or\_not\_list

calculating\_who\_is\_priority(priority\_list):

LOOP through patient in priority\_list

Count = 0

*# uses DeMorgan's law to provide specific Boolean statements  
# which can be written in different ways for the same effect.*IF (not (not ((patient.Asthmatic or patient.Smoker)  
 and patient.patients\_age > 55) and not (patient.class\_of\_patients\_weight == "Obese "and patient.Hypertension))):  
 count += 3 *# this will be counted if patients has all 3*

IF patient is smoker

Increment is added by 1

IF patient is asthmatic

Increment is added by 1

IF patient has condition of NJT or NGR

Increment is added by 1

IF patient has hypertension

Increment is added by 1

IF patient has Renal\_RT

Increment is added by 1

IF patient has Ileostomy or Colostomy

Increment is added by 1

If patient has parenteral nutrition

Increment is added by 1

If the count is more than 3

SET the patient as top priority and set it as true

RETURN priority\_list

top\_priority\_or\_not():

CALL read\_csv to create a list data structure containing objects of the patient class called patient\_list

LOOP through patient in patient\_list

SET\_class weight\_of\_patient

CALL patient\_list to store all the patient’s top priority and attributes into calculating\_who\_is \_priority

CREATE empty list called list\_of\_patient\_who\_is\_top\_priority

LOOP patient in patient\_list

IF patient is top\_priority

REMOVE from patient\_list

APPEND to list\_of\_patient\_who\_is\_top\_priority

*Sorts out the patients by their priority using lambda expression and the reverse method –*

*patient\_list.sort(reverse=True, key=lambda patient: patient.topPriority)  
list\_of\_patient\_who\_is\_top\_priority.sort(reverse=True, key=lambda patient: patient.topPriority)*

CREATE variable called top\_priority\_list from list\_of\_patient\_who\_is\_top\_priority + patient\_list

Count = 0

LOOP through patient in top\_priority\_list

CALL patient\_fields\_task2

Increment is added by 1

IF count is more than 10

ADD INPUT ("\n Please press enter to proceed for the next 10 patients. \n")

CALL top\_priority\_or\_not()

**Diagrams for task 2**

Diagram

Description automatically generated

Diagram

Description automatically generated