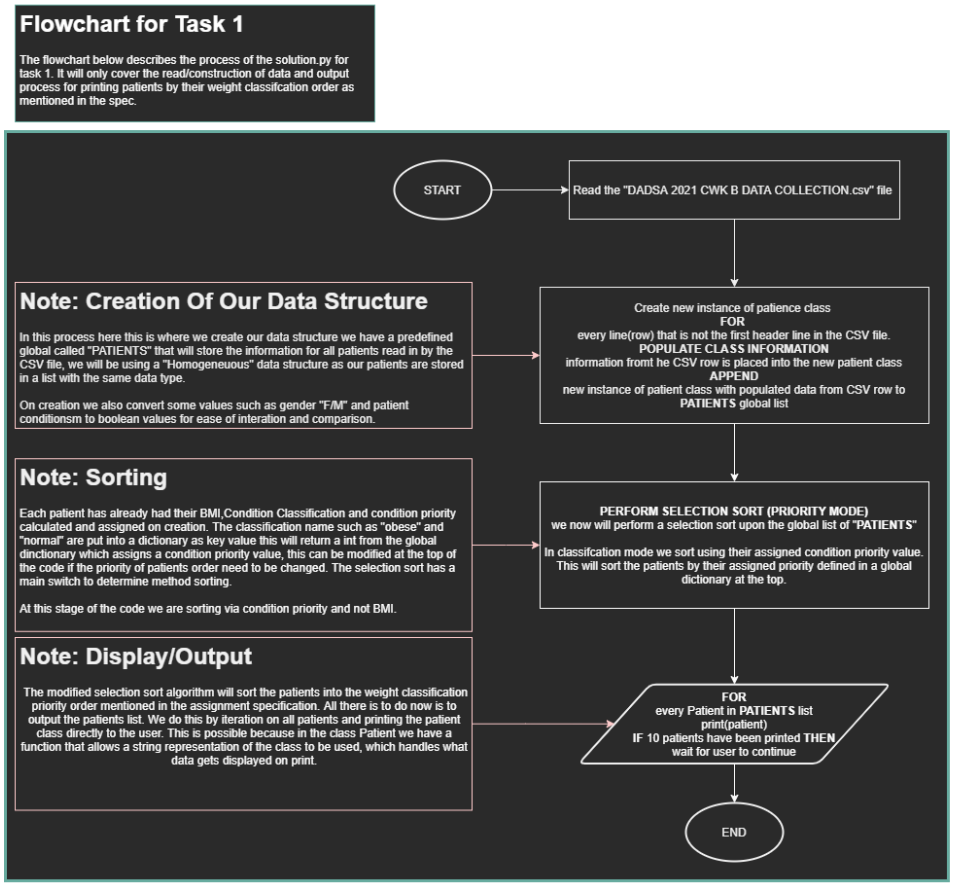
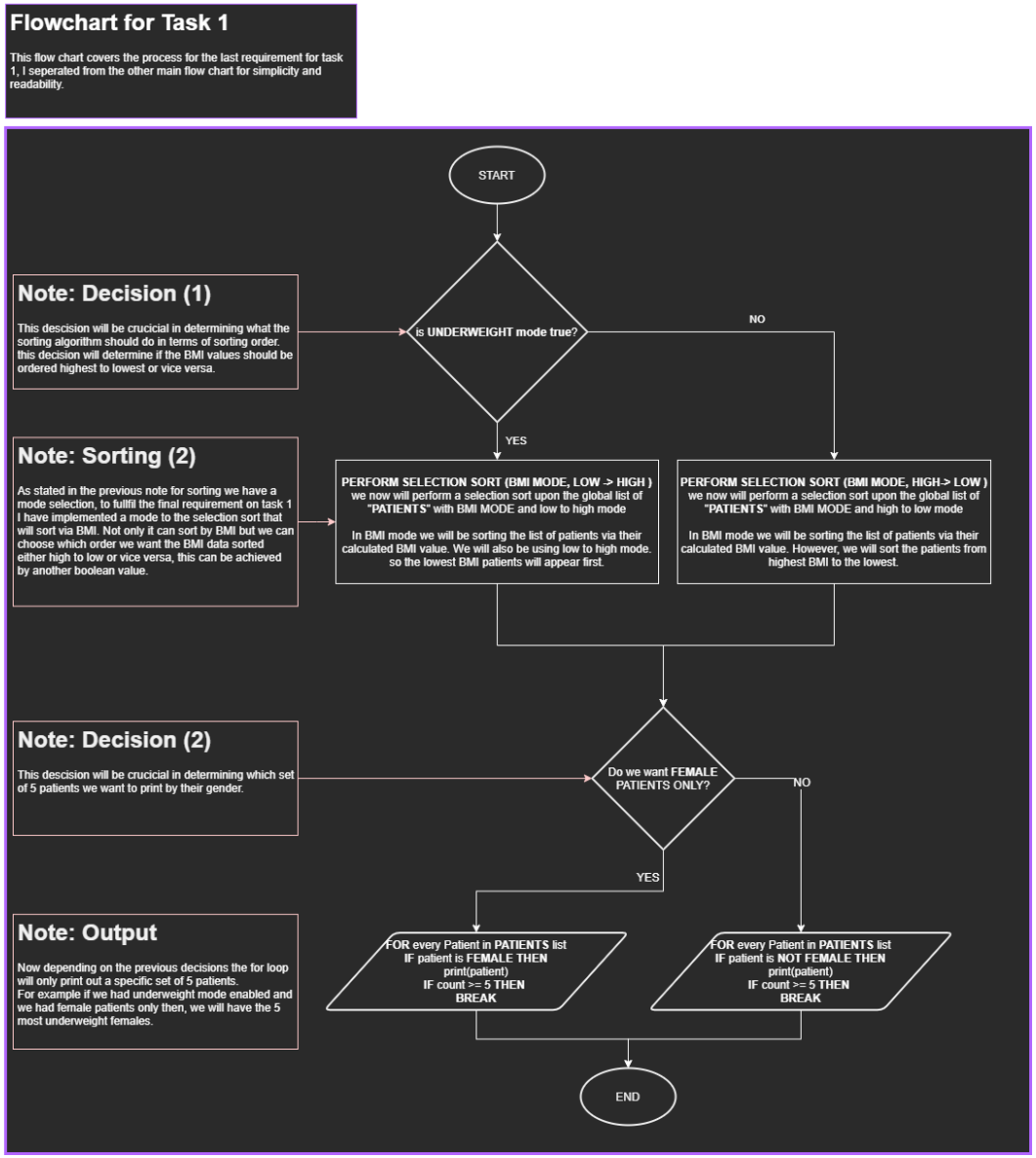
FLOWCHART DESIGN DIAGRAM FOR TASK 1





In the flowchart diagram above you can see my design for task 1. In this example I have described and outlined the processes of how my python script will handle and sort the patient information data and from the CSV file.

PSEUDO CODE

Read Datafile

```
FUNCTION READ_DATAFILE(): called on application start when we want to read in the csv data

    global variable reference to patients
    global PATIENTS

    open the datafile and access it via variable named csvfile
    WITH open(datafile_name()) AS csvfile:
        line_count = 0 tracks the lines

        csv reader will read the csv file and separate it with the given delimiter character of a comma
        spamreader = csv.reader(csvfile, delimiter=',')

        FOR row IN spamreader:  for every row in the CSV
            IF line_count != 0: IGNORE FIRST LINE OF CSV

                Create new patient and argument pass the row data into the patient object.
                row[0] will be patient name, row[1] will be patient date of birth.
                Then append that patient to the GLOBAL patients list referenced at the top of this function

            line_count += 1 increment line count, used to track which line to ignore

        close the CSV file once we done reading it
        csvfile.close()


    Perform selection sort with no additional arguments, by default the patients will be sorted by
    condition priority which is determined and calculated on creation of new patient within the
    constructor
        selection_sort(PATIENTS)
```

In the provided text above you can see my pseudo code for task 1. In this example I am showing the read datafile method. This segment of code is responsible for reading the CSV file and converting it into a homogenous data structure. Each row of the CSV file will have its column values passed into a new instance of patient class and its data simplified to the use of Ternary operators: “Y” == True etc.

Patient Class

```
CLASS PATIENT:

    FUNCTION CONSTRUCTOR( self, name, dob, isfemale, height, weight, bodybuild, smoker, asthmatic, NJT, hypertension, renal_rt, ileostomy, parenteral_nutrition ):

        On class construction store passed argument values on this class self

        self.identity_name = id_name
        self.dob = CONVERT STRING DOB TO DATETIME DATA TYPE
        self.is_female = is_female
        self.height = height
        self.weight = weight
        self.body_build = body_build
        self.smoker = smoker
        self.asthmatic = asthmatic
        self.nasogastric_tube = nasogastric_tube
        self.hypertension = hypertension
        self.renal_rt = renal_rt
        self.ileostomy = ileostomy
        self.parenteral_nutrition = parenteral_nutrition

        Calculate BMI and BMI Classification and priority on patient creation, as well as their age and assign it to self in the class
        self.bmi = CALCULATE_BODYMASS_INDEX(self)
        self.classification = DETERMINE_CLASSIFICATION(self)
        self.conditionPriority = CONVERT BMI CLASSIFICATION TO PRIORITY
        self.age = CALCULATE_DOB_TO_AGE(self)
```

In the provided text above you can see my pseudo code for task 1. In this example I am showing the patient class. This class is responsible for holding all data about our patient.

Sorting Algorithm: Selection

```
# selection_sort :: takes in python list of patient objects sorts them by condition priority if sort_via_bmi is false
# if sort via bmi is true then low_to_high determines the sorting order of the BMI values
FUNCTION SELECTION_SORT(array, sort_via_bmi = False, low_to_high = True):

    FOR i IN range(0, len(array)): # for every element in the list named array
        Finding the smallest number in the subarray
        min = i

        for every element in the array from i + 1 we compare the condition values if sort via bmi Boolean is false
        FOR j IN range(i+1,len(array)):

            IF sort_via_bmi: sorts via BMI if argument is provided, default is false
                sorts the BMI low to high by default if set to false the ternary operators will switch the sort order to high to low
                IF ((array[min].bmi > array[j].bmi) IF low_to_high ELSE (array[min].bmi < array[j].bmi) ):
                    min = j # set min from j index found
            ELSE:
                if array[min].conditionPriority > array[j].conditionPriority: # conditionPriority is a int value set on creation of patient
                    set min from j index found
                    min = j

        IF (min != i):
            # perform swap
            temp = array[i]
            array[i] = array[min]
            array[min] = temp
```

In the provided text above you can see my pseudo code for task 1. In this example I am showing the sorting algorithm used to sort the patients. The sorting algorithm will sort the patients via lowest to highest by default unless the Boolean is given to sort it highest to lowest. There is also another Boolean that determines if we should sort via bmi values or their assigned given condition priority.

Calculate Body Mass Index Function

```
# calculate_bodymass_index :: takes in object type Patient, gets weight and height from patient object
# returns float body mass index
FUNCTION CALCULATE_BODYMASS_INDEX(patient):
    RETURN patient.weight / patient.height**2
```

In the provided text above you can see my pseudo code for task 1. In this example I am showing the calculate body mass index function. This function is responsible for calculating the BMI of a given patient object, the patient object will store their weight and height as a public variable. We access these variables to calculate the BMI with the given equation in the spec. the result is then assigned upon the patient object as a new value called BMI.

Determine patient classification Function:

```
determine_classification :: takes in object type Patient::returns string classification
FUNCTION DETERMINE_CLASSIFICATION(patient):

    depending on the body build we change the overweight_to_obese_threshold :: seems to be only factor changing
    IF patient.body_build.lower() == "slim":
        overweight_to_obese_threshold = 28
    ELIF patient.body_build.lower() == "regular":
       overweight_to_obese_threshold = 29
    ELIF patient.body_build.lower() == "athletic":
        overweight_to_obese_threshold = 30

    depending on the patient BMI and the overweight_to_obese_threshold
    return the classification of this patient which is assigned in patient class
    IF(patient.bmi < 18.5):
        RETURN "underweight"
    ELIF(patient.bmi > 18.5 and patient.bmi < 25):
        RETURN "normal"
    ELIF(patient.bmi > 25 and patient.bmi < overweight_to_obese_threshold):
        RETURN "overweight"
    ELIF(patient.bmi > overweight_to_obese_threshold):
        RETURN "obese"
```

In the provided text above you can see my pseudo code for task 1. In this example I am showing the determine patient classification. This function is responsible for calculating and assigning a patient classification name. The patient object will have variables stored upon it that contain their body build and bmi. We access these variables to calculate the patient classification to the requirements of the spec. the classification name is then returned and assigned on the patient object.

Calculate Age

```
calculate_dob_to_age :: takes in object type Patient :: returns int age of patient
FUNCTION CALCULATE_DOB_TO_AGE(patient):
    get the current date now
    get the patient born date of birth

    calculate the difference from datenow and the born date from the patient :: return int year
```

In the provided text above you can see my pseudo code for task 1. In this example I am showing the calculate dob to age function. This function is responsible for calculating and assigning a patient age. The patient object will have a variable stored upon it that contain date of birth. We access this variable to calculate the patient age by finding the difference with date time now. The returned value is assigned and stored on the patient.