# CS 1026 B, Winter 2023 Assignment 3

# Creating an Information System for Electronic Medical Records

*Due: ~~Wed Mar 22, 23:55pm~~ via Gradescope*. **Extended to give you more time and flexibility to Tues Mar 28, 23:55pm**

Changelog and general notices:

- 2023-03-17:
  - *Function 1 was updated to have the same "Errors to Handle" as Function 4*
- 2023-03-21:
  - *In Function 4, further clarification was added about valid date ranges. **Only a simple check is required**, so DD must be in the range 1-31, MM must be in range 1-12, and year should be 1900 or more recent. Future dates are allowed. You do not need to add more advanced logic for the number of days per specific month (you don't need to check if Feb is <= 28 days, or handle leap years, etc.)*

- 2023-03-22:
  - *Please remember you do not need to write your own main.py file from scratch. Please download the template main.py file in Section 4 and fill in your own code where it's indicated to make the functions work*
- 2023-03-22:
  - *As per a number of student requests, we are extending A3 deadline to give you the weekend. It is now due Tuesday 2023-03-28 23:55pm. Hope this reduces your stress at this busy time of year!!*
- 2023-03-28 - Some clarifications based on recent student questions:
  - *In Function 4 - addPatientData, the in the requirements, it says to print "**Visit is saved successfully for Patient #<patientId>**", but in the test case screenshot, it prints: "Visit saved for Patient # 4." . Please use the wording provided in the assignment text and not the screenshot.*
  - *In Function 6 - findPatientsWhoNeedFollowUp(), if the same patient has multiple abnormal vital signs, they only need to be added a single time. Let's assume once a doctor meets this patient, they will look at all of that patient's record.*
  - *In Function 4 - addPatientData, the code should accept all inputs from the user, then run the checks in the order presented in the assignment. It will then stop on the first error it finds.*
  - *In Function 3 - displayStats, it is unclear about whether to include 0, or 1, or 2 spaces. For this function, Autograder is set to be permissive, so it will accept both with 0, 1, or 2 spaces*.

**Table of Contents** ☰

## 1) Background

In Canada, we are seeing the need to improve the healthcare system. What many people don't realise is that our healthcare system still makes use of very primitive information systems like excel sheets, fax machines, and even paper written records. There is a ton of room for improvements that we as Computer Scientists can make.

Image: Health Tech and Interface

For example, Electronic Medical Records (EMRs) are becoming increasingly important in the healthcare industry, providing a way for doctors and other healthcare professionals to access a patient's medical history, diagnosis, and treatment plans all in one place. By automating the process of collecting, storing, and analyzing patient data, EMRs help to improve the quality of care, reduce errors, and increase patient safety.

## Assignment Intro Video

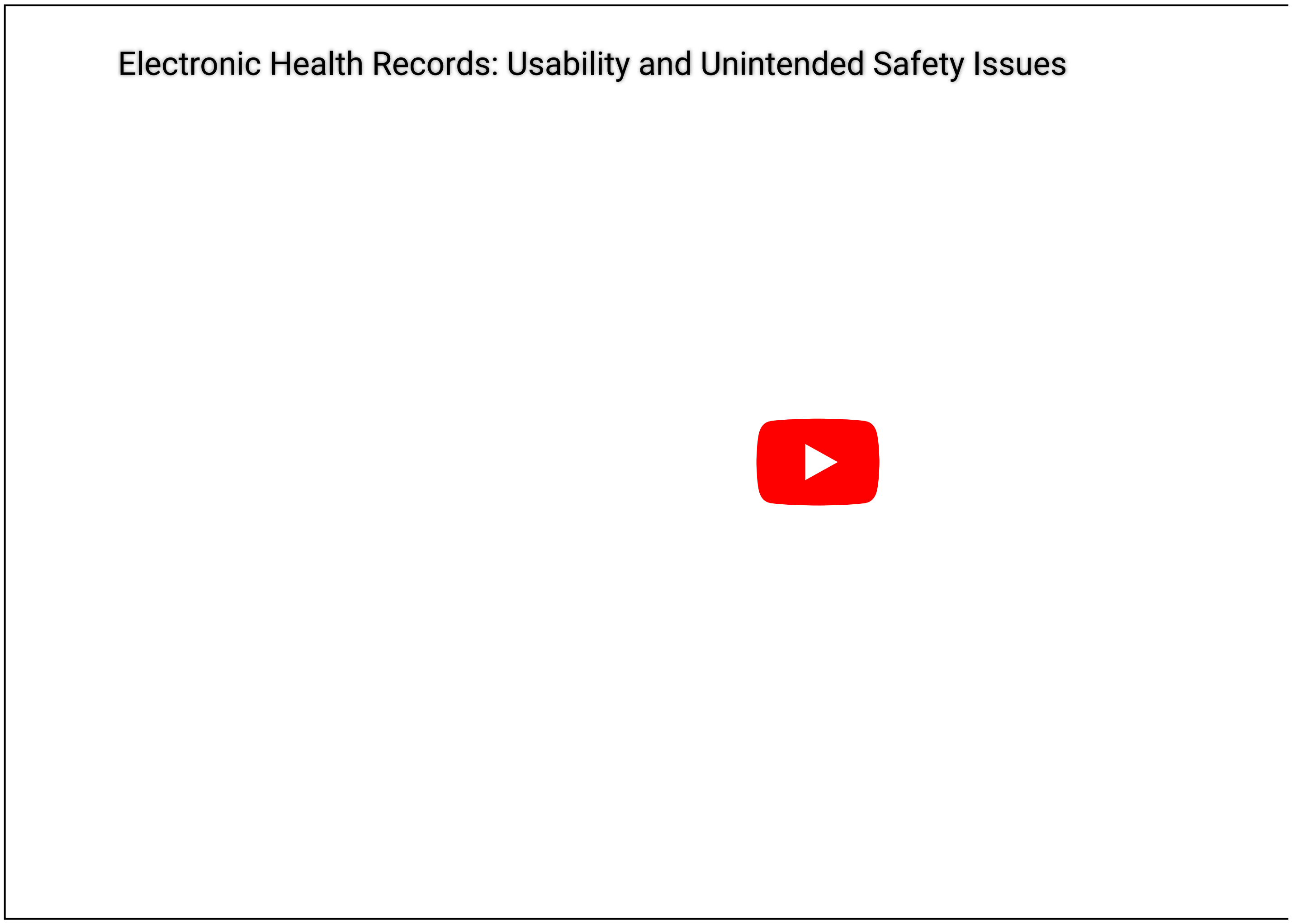
This video is required to watch as it summarises the assignment.

https://youtube.com/watch?v=DBC89rs_9I8&si=EnSIkaIECMiOmarE

Brief Video to Introduce the Assignment

## Context of Assignment and Future Applications

The following video provides some context on the benefit and impacts of design of software used in hospitals. This video is optional, but recommended to understand the issues in health information systems.


Electronic Health Records: Usability and Unintended Safety Issues

Context Video: Issues with Health Information Systems

## Scope and Context

In this programming assignment, you will be creating a Health Information System that uses biomedical signals to produce an electronic medical record for each patient. The system will read patient data from a plain-text file that is separated by commas, allowing new patient information to be added through user input to update the plain-text file..

In programming, the term "backend" refers to the part of a software system or application that is responsible for handling data processing, and other functionality that is not directly visible to the user. This includes database management, and other aspects of a system that do not involve the user interface (UI). The backend is typically responsible for processing requests and delivering data to the frontend (user-facing part of the software) for display to the user. It is often designed to be scalable, flexible, and able to handle large volumes of data efficiently.

Though we will not be focusing on User-Interface design (UXD) in this assignment, your work in this assignment will provide a possible core information system “backend” to help improve a healthcare system.

## Assignment Walk-Through Video by Prof (Required)

https://youtu.be/ENaKj1crDVA

Thank you for your help in improving the healthcare system!

## 2) Program Features

The program will provide the following features, plus more as described in the function sections below:

- A function to read patient data from a plaintext file.
- A function to display various statistics about the patients, such as the average vital signs about the patients or for a specific patient.
- A function to display patient data for all patients or for a specific patient by ID.
- A function to add new patient data to the patient list.
- A function to find patient visits in a certain month or year
- A function to find patients who need follow-ups depending on their vital statistics
- A text-based interface to list the patients and allow the user to select which feature they would like to use.

## 3) Dataset

We will use a text file containing data from patient visits in a file, called “patients.txt”. The file contains one line for every unique visit by any patient to the clinic.  Note that it is possible that each patient may visit the clinic multiple times. The file has the following columns, with each data element being separated with a comma and each new record on a new line.

The data in the file will look like the following:

```
1,2022-06-01,37.2,72,16,120,80,97
1,2022-09-15,37.5,73,18,125,82,96
2,2022-06-10,36.9,69,18,123,80,96
2,2023-01-03,37.1,71,19,124,81,97
3,2022-08-05,37.3,74,20,126,83,95
3,2023-03-01,37.4,75,18,123,79,98
```

This data will be structured with the delimiter being a comma, and the types of information as follows:

- Patient ID
    - Integer
- Date of visit (YYYY-MM-DD)
    - String
- Body temperature (Celsius)
    - Float
- Heart rate (beats per minute)
    - Integer
- Respiratory rate (breaths per minute)
    - Integer
- Systolic blood pressure (mmHg)
    - Integer
- Diastolic blood pressure (mmHg)
    - Integer
- Oxygen saturation (%)
    - Integer

*A full sample dataset is available online with the .txt extension. For testing purposes, you are strongly encouraged to create your own datasets following the above convention to test your code more thoroughly.*

## 4) File to Hand In and Provided Files

You are provided a sample main.py file with indications where you'll put your own implementation. You will submit one file – named "main.py" to Gradescope. Please do not modify the main function in this file.

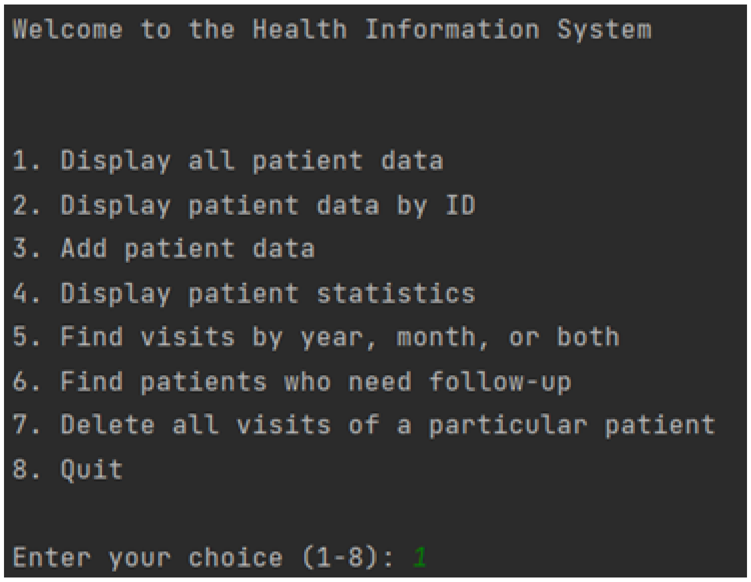
## Links to Provided Files:

- main.py (https://1drv.ms/u/s!As0vrM-NsbcQkpYwuuCIP0JbvqxyJQ?e=eAK5SM)
  - This file contains the definitions of the functions you will write, as well as the main program the creates the interface.
  - You will write code to implement these functions as per the specifications at the location indicated by a comment "#Put your code here".
- patients.txt (https://1drv.ms/t/s!As0vrM-NsbcQkpYvHATJ4v-zz3CIcA?e=LSHQJi)
  - This is one sample dataset containing visits of patients and their medical data. You can use this for testing. We encourage you to modify it and test with different data.

## 5) Learning Outcomes and Skills Used

- **Using functions**: Consider the use of functions to break down the tasks into smaller, more manageable parts. E.g., implement functions for reading patient data from a file, displaying patient data, adding new patient data, and calculating patient statistics.
- **Complex data structures**: Use complex data structures, such as dictionaries and lists, to store and manipulate patient data.
- **Text processing**: We will use text processing to read and write patient data to and from a text file.
- **File input and output**: We will implement functions to read and write patient data to a text file.
- **Exceptions in Python:** Consider handling exceptions in our code. For example, consider how to handle cases where the data in the text file is not in the correct format, or has an extra blank line at the end.
- **Using Python modules**: We should use Python modules as applicable, to make the code more efficient and readable.
- **Adhering to specifications by testing programs and developing test cases**: Remember to adhere to the specifications given and test the programs using test cases.

## 6) View of Information System Interface you will Create



```
Welcome to the Health Information System


1. Display all patient data
2. Display patient data by ID
3. Add patient data
4. Display patient statistics
5. Find visits by year, month, or both
6. Find patients who need follow-up
7. Delete all visits of a particular patient
8. Quit

Enter your choice (1-8): 1
```

Sample View from User's Perspective

## 7) General Tips

- Break down the tasks into smaller, more manageable parts. This will make the code easier to write and easier to maintain.
- Use meaningful variable and function names. This will make the code easier to read and understand. Function names are provided.
- Use docstrings to document the functions. This will make the code easier to understand and maintain. Docstrings are provided.
- Write detailed comments in the code. Not only is this part of your grade for the assignment, but it's building a best-practice for when you are doing coding projects with others after this course is over.
- Test the code using your own test cases. This will help identify and fix bugs in the code. Feel free to make other input files, and plan out your user inputs to see if code works as expected.

## 8) Starting

To help you get started, we have provided function declarations and docstrings for each of the above features. We have also included tips for each function to help you implement them effectively. Remember to test your code using test cases to make sure it is working as expected. Good luck!

## 9) Function 1: readPatientsFromFile()

*Note the following correction made to Function 1 requirements*

Correction summary: number ranges in "Errors to Handle" section - made on 2023-03-17

Correction Details: Old Vs New "Errors to Handle" section to make Function 1 be consistent with Function 4

This function reads patient data from a plaintext file and returns a dictionary, returns a dictionary where each key is a unique patient ID and the corresponding values are 2-dimensional lists storing information about each visit (where each visit's information is stored in its own inner-list)

```python
def readPatientsFromFile(fileName):
    """
    Reads patient data from a plaintext file.

    fileName: The name of the file to read patient data from.
    Returns a dictionary of patient IDs, where each patient has a list of
    The dictionary has the following structure:
    {
        patientId (int): [
            [date (str), temperature (float), heart rate (int), respirator
            [date (str), temperature (float), heart rate (int), respirator
            ...
        ],
        patientId (int): [
            [date (str), temperature (float), heart rate (int), respirator
            ...
        ],
        ...
    }
    """
```

## Functionality

- Reads patient data from a plaintext file specified by the **fileName** argument.
- Returns a dictionary of patient IDs, where each patient has a list of visits.
- Each visit is a list of 7 values in the order of date, temperature, heart rate, respiratory rate, systolic blood pressure, diastolic blood pressure, and oxygen saturation, in this order.
- Performs additional data validation checks to ensure the data is within reasonable ranges (as listed below)
- If any of the data is invalid or an error occurs (the list of errors is listed below), the function skips the line that contains the error, and continues reading the next line in the file.

## Errors to Handle

*Changelog: Please note that the ranges in this section were corrected to match the ranges in Function 4 on Mar 17.*

- If the file specified by **fileName** cannot be found, print the message: "The file '[fileName]' could not be found." and the program exits.
    - You can assume that you will be passed a file name and not a path.
- If there is an unexpected error while reading the file, print the message: "An unexpected error occurred while reading the file." And skips to the next line.
- If the number of fields in a line is not equal to 8, print the message: "Invalid number of fields ([numFields]) in line: [line]" And skips to the next line
- If the patient ID or any of the other 7 data values are not in the correct format, print the message: "Invalid data type in line: [line]" And skips to the next line
- If the temperature value is not within the range of 35 to 42, print the message: "Invalid temperature value ([temp]) in line: [line]" And skips to the next line
- If the heart rate value is not within the range of 30 to 180, print the message: "Invalid heart rate value ([hr]) in line: [line]" And skips to the next line
- If the respiratory rate value is not within the range of 5 to 40, print the message: "Invalid respiratory rate value ([rr]) in line: [line]" And skips to the next line
- If the systolic blood pressure value is not within the range of 70 to 200, print the message: "Invalid systolic blood pressure value ([sbp]) in line: [line]" And skips to the next line

- If the diastolic blood pressure value is not within the range of 40 to 120, print the message: "Invalid diastolic blood pressure value ([dbp]) in line: [line]"
- If the oxygen saturation value is not within the range of 70 to 100, print the message: "Invalid oxygen saturation value ([spo2]) in line: [line]" And skips to the next line

## Exact Output Formatting

- Each error message should be printed exactly as specified in the requirements, with the appropriate variable values filled in.
- if any of the error messages are printed, we do not add this visit to the dictionary at all, and we move on to the next line

## What should the dictionary contain?

If the input file looked like this:

```
1,2022-06-01,37.2,72,16,120,80,97
1,2022-09-15,37.5,73,18,125,82,96
2,2022-06-10,36.9,69,18,123,80,96
2,2023-01-03,37.1,71,19,124,81,97
3,2022-08-05,37.3,74,20,126,83,95
3,2023-03-01,37.4,75,18,123,79,98
```

Your dictionary would look like this:

```
{1: [['2022-06-01', 37.2, 72, 16, 120, 80, 97], ['2022-09-15', 37.5, 73, 18
```

## Test Cases

Case 1: Information invalid

Case 2: Information valid and successful import

*For examples on how to test other inputs, please watch the extended assignment intro video.*

# 10) Function 2: displayPatientData()

This function displays patient data for all patients or a given patient ID.

```python
def displayPatientData(patients, patientId=0):
    """
    Displays patient data for a given patient ID.

    patients: A dictionary of patient dictionaries, where each patient has
    patientId: The ID of the patient to display data for. If 0, data for al
    """
```

## Functionality

- The function takes in a dictionary **patients** containing patient data where each patient has a list of visits, and a **patientId** (default 0) which represents the ID of the patient to display data for. If **patientId** is 0, then the function displays data for all patients.
- The function prints patient data for the specified **patientId** or for all patients if **patientId** is 0. For each patient, the function prints the patient ID and data for each visit including visit date, temperature, heart rate, respiratory rate, systolic blood pressure, diastolic blood pressure, and oxygen saturation.

## Errors to handle

- If **patientId** is not found in **patients**, the function should print an error message "Patient with ID {patientId} not found." And the program will go back to the main menu
- Additional error checking (e.g, checking if patients are not a dictionary, or patient ID is negative or not an integer, etc.) is not necessary in this function.

## Exact output required

- For each patient, the function prints the patient ID followed by data for each visit. For each visit, the function prints:
  - Visit Date: {visit date in yyyy-mm-dd format}
    - Append 1 space to start of each visit so it appears indented from patient ID, e.g.

```python
print(" Visit Date:", visit[0])
```

- - Temperature: {temperature value} C
    - Append 2 spaces to each health stat so it appears indented from visit date, e.g.
  -

```
print("  Temperature:", "%.2f" % visit[1], "C")
```

- - Heart Rate: {heart rate value} bpm
  - Respiratory Rate: {respiratory rate value} bpm
  - Systolic Blood Pressure: {systolic blood pressure value} mmHg
  - Diastolic Blood Pressure: {diastolic blood pressure value} mmHg
  - Oxygen Saturation: {oxygen saturation value} %
- The function should not return anything.

## Test cases

1) Print all cases

2) Patient is not in data

3) Patient is found in data

## 11) Function 3: displayStats()

This function prints the average of each vital sign for all patients or for a specified patient.

```
def displayStats(patients, patientId=0):
    """
    Prints the average of each vital sign for all patients or for the speci

    patients: A dictionary of patient IDs, where each patient has a list of
    patientId: The ID of the patient to display vital signs for. If 0, vita
    """
```

## Functionality:

- The function should display the average vital signs of all patients or a specific patient, depending on the value of the **patientId** parameter (which should default to 0).
  - If **patientId** is 0, the function should display the average of each vital sign for all patients contained in the data, as a single average (not per patient).
  - If **patientId** is a positive integer, the function should display the average vital signs for the specified patient.
  - If the specified **patientId** is not found in the **patients** dictionary, an error message **print("No data found for patient with ID {}.")** should be displayed, and the program will go back to the main menu.
- The function should display the following vital signs with their respective units:
  - Average temperature in degrees Celsius (°C)
  - Average heart rate in beats per minute (bpm)
  - Average respiratory rate in breaths per minute (bpm)
  - Average systolic blood pressure in millimeters of mercury (mmHg)
  - Average diastolic blood pressure in millimeters of mercury (mmHg)
  - Average oxygen saturation as a percentage (%)
  - If no data is found for the specified patient or for all patients, a message indicating that no data was found should be displayed (as above), and the program will go back to the main menu.

## Errors to handle

- If **patientId** is not an integer, an error message should be displayed,
  - print("**Error: 'patientId' should be an integer.**") and then return to the main menu
- If the **patients** parameter is not a dictionary, an error message should be displayed:
  - Print ("**Error: 'patients' should be a dictionary**.") and then return to the main menu

## Output format

- The function should display the vital sign values rounded to exactly two decimal places of precision.
- The function should display each vital sign on a separate line with a specific message structure, as follows:

- If **patientId** is 0, the message should start with "Vital Signs for All Patients:" followed by the average vital sign value and unit.
- If **patientId** is a positive integer, the message should start with "Vital Signs for Patient {patientId}:" followed by the average vital sign value and unit.
- The vital sign should have 2 spaces inserted so it appears tabbed in from the line "Vital signs for Patient …" e.g.

```
print("  Average temperature:", "%.2f" % (temp_sum / num_visits), "C")
```

If no data is found for the specified patient or for all patients, the function should display a message with the structure "No data found" and the program will go back to the main menu.

## Test Cases

1) Patient ID is not an integer

2) Patient ID is 0

3) Patient ID is positive integer

4) Patient ID is not found in data

## 12) Function 4: addPatientData()

This function adds new patient data to the patients dictionary **and** appends this same data to the text file.

```
def addPatientData(patients, patientId, date, temp, hr, rr, sbp, dbp, spo2
    """
    Adds new patient data to the patient list by appending to the dictiona

    patients: The dictionary of patient IDs, where each patient has a list
    patientId: The ID of the patient to add data for.
    date: The date of the patient visit in the format 'yyyy-mm-dd'.
    temp: The patient's body temperature.
    hr: The patient's heart rate.
    rr: The patient's respiratory rate.
    sbp: The patient's systolic blood pressure.
    dbp: The patient's diastolic blood pressure.
    spo2: The patient's oxygen saturation level.
    fileName: The name of the file to append new data to.
    """
```

## Requirements

Requirements for the implementation of **addPatientData()** function are:

- The function should take in a dictionary of patient IDs as **patients**, where each patient ID has a list of visits.
- The function should take in the following parameters for adding new patient data: **patientId**, **date**, **temp**, **hr**, **rr**, **sbp**, **dbp**, and **spo2**.
- The function should take in a file name as **fileName** and append the new data to this file.
  - You can assume that we will use the same text file to write to as was read in using the readPatientsFromFile() function
- The function should check the inputs and handle erroneous input as below.
- The function should add the new data to the patient's visit history by appending the visit information to the text file.
- The function should display an error message if any input value is invalid and return to the main menu without adding the new data.

## Errors to Handle

*Edit note - this section updated 2023-03-15 to clarify the errors.*

If any of the following errors occur, the program should display the exact corresponding message and return to the main menu

- If the date format is invalid, print “Invalid date format. Please enter date in the format ‘yyyy-mm-dd’.”
- If the date is cannot exist, e.g., the 35th day of a month or a 14th month, etc, print "Invalid date. Please enter a valid date."
    - *Further clarification 2023-03-21- **Only a simple check is required**, so DD must be in the range 1-31, MM must be in range 1-12, and year should be 1900 or more recent. Future dates are allowed. You do not need to add more advanced logic for the number of days per specific month (you don't need to check if Feb is <= 28 days, or handle leap years, etc.)*
- If the temperature is invalid, print "Invalid temperature. Please enter a temperature between 35.0 and 42.0 Celsius."
- If the heart rate is invalid, print "Invalid heart rate. Please enter a heart rate between 30 and 180 bpm."
- If the respiratory rate is invalid, print "Invalid respiratory rate. Please enter a respiratory rate between 5 and 40 bpm."
- If the systolic blood pressure is invalid, print "Invalid systolic blood pressure. Please enter a systolic blood pressure between 70 and 200 mmHg."
- If the diastolic blood pressure is invalid, print "Invalid diastolic blood pressure. Please enter a diastolic blood pressure between 40 and 120 mmHg."
- If the oxygen saturation level is invalid, print "Invalid oxygen saturation. Please enter an oxygen saturation between 70 and 100%."
- If an unexpected error occurs (any exception), print "An unexpected error occurred while adding new data."

## Exact Output Required

The function should display a success message after adding the new data. If the new data is successfully added, print "**Visit is saved successfully for Patient #<patientId>**", and return to the main menu

## Test Cases

1) Invalid Date

2) Out of range health stat (oxygen saturation) ›

3) Valid input ›

## 13) Function 5: findVisitsByDate()

This function finds the visits by date provided and returns a list of tuples. See below.

```
def findVisitsByDate(patients, year=None, month=None):
    """
    Find visits by year, month, or both.

    patients: A dictionary of patient IDs, where each patient has a list of
    year: The year to filter by.
    month: The month to filter by.
    return: A list of tuples containing patient ID and visit that match the
    """
```

### Functionality:

- The function should take in a dictionary of patient IDs, where each patient has a list of visits, as well as an optional year and/or month to filter the visits by.
- The function can accept a year only, or both a year and a month. It can't accept a month only:
  - If the year is only provided, the function returns all visits occurred in that year regardless of the month and day.
  - If the month and year are provided, the function returns all visits occurred in that month of the provided year regardless of the day.
  - If no year and/or month is provided, the function should return all visits for all patients.
- The function should ignore visits in the dataset with incomplete or invalid date information (i.e. dates that are missing year, month, or day).
- For example, if the text file contained:

```
1,2022-06-01,37.2,72,16,120,80,97
1,2022-09-15,37.5,73,18,125,82,96
2,2022-07-10,36.9,69,18,123,80,96
2,2023-01-03,37.1,71,19,124,81,97
3,2022-08-05,37.3,74,20,126,83,95
3,2023-03-01,37.4,75,18,123,79,98
```

And month: 6 and year 2022 was provided, your function should return this:

```
[(1, ['2022-06-01', 37.2, 72, 16, 120, 80, 97]), (2, ['2022-06-10', 36.9, 6
```

## Errors to Handle

- The function should handle the case where the patient dictionary is empty, or where the patient dictionary is not empty, but there are no visits in the given year/month . In these cases, your program should return an empty list.
- The function should handle the case where an invalid year or month is provided (e.g, the 14th month or 35th day, etc) by ignoring the filter and returning an empty list.

## Output Format

- The function should return a list of tuples where each tuple contains the patient ID (an integer) and visit (a list).
- The visit list should contain the following information in this order: date (a string in the format 'yyyy-mm-dd'), temperature (a float), heart rate (an integer), respiratory rate (an integer), systolic blood pressure (an integer), diastolic blood pressure (an integer), and oxygen saturation (an integer).

## Test Cases

1. Invalid date
2. Input just a year
3. Input a month and year (data does not exist)
4. Input a month and year (data exists)

## 14) Function 6: findPatientsWhoNeedFollowUp()

```
def findPatientsWhoNeedFollowUp(patients):
    """
    Find patients who need follow-up visits based on abnormal vital signs.

    patients: A dictionary of patient IDs, where each patient has a list of
    return: A list of patient IDs that need follow-up visits due to abnorma
    """
```

### Functionality

- The function takes in a dictionary of patient IDs and their visits as input.
- It checks each visit of each patient for abnormal vital signs (heart rate, systolic and diastolic blood pressure, and oxygen saturation level) using if conditions.
- If any visit has any of the abnormal vital signs, the patient's ID is added to the followup_patients list.
- For the assignment, we'll assume if **any** of the following are true, the patient needs a followup
  - Heart rate > 100 or heart rate is < 60 or systolic is > 140 or diastolic is > 90 or blood oxygen saturation is < 90:
- The function returns a list of patient IDs that need follow-up visits.
  - If no patients require a follow-up, return an empty list.

**Errors to handle**

- None.

**Exact output required**

- The function should return a list of integers representing the patient IDs that need follow-up visits.
  - Note that the main program provided will handle printing this list out for you.
  - For debugging/testing purposes, consider printing out the contents of your list, then removing this print statement before submitting.

## Test Cases

1. Case At least 1 health metric is cause for a followup
2. Case No patients need followup

## 15) Function 7: deleteAllVisitsOfPatient()

```
def deleteAllVisitsOfPatient(patients, patientId, filename):
    """
    Delete all visits of a particular patient.

    patients: The dictionary of patient IDs, where each patient has a list
    patientId: The ID of the patient to delete data for.
    filename: The name of the file to save the updated patient data.
    return: None
    """
```

### Functionality:

- The function **deleteAllVisitsOfPatient** deletes all visits of a particular patient from the **patients** dictionary.
- It rewrites the text file with the updated patient data.

#### Arguments:

- **patients**: A dictionary of patient IDs, where each patient has a list of visits.
- **patientId**: The ID of the patient to delete data for.
- **filename**: The name of the file to save the updated patient data.

#### Returns:

- None

#### Errors to handle:

- If **patientId** is not found in **patients**, print **"No data found for patient with ID {patientId}"**.

Output:

- If data for the patient with **patientId** is deleted, print "Data for patient {patientId} has been deleted.".
- No output if **patientId** is not found in **patients**.

## File Writing Requirements:

1. The function should open the specified file in write mode (**'w'**).
    - It is important to note that the w mode will overwrite the data currently in the file, but you will need to write all remaining patients and visits back to the file.
    - When testing this function, your code will actually delete the records from the text file. You will need to re-download the text file to “reset” for your next test.
2. The function should iterate over the dictionary of patient data and write each remaining visit for each patient to the file in comma-separated format **after removing the deleted patients visit(s)** . The format for each line in the file should be:

```
patientId,date,temp,hr,rr,sbp,dbp,spo2\n
```

where **patientId** is the ID of the patient, **date** is the date of the visit in the format **yyyy-mm-dd**, **temp** is the patient's body temperature, **hr** is the patient's heart rate, **rr** is the patient's respiratory rate, **sbp** is the patient's systolic blood pressure, **dbp** is the patient's diastolic blood pressure, and **spo2** is the patient's oxygen saturation level.

Note that there is no space before or after the commas, and each line should end with a newline character (**\n**).

3. After writing all the visits to the file, the function should close the file.

## Test Cases

1. Case patient exists

2. Case patient does not exist

## 16) Marking Guidelines

1. The assignment will be marked as a combination of your auto-graded test results, manual grading of your code logic, comments, formatting, style, etc. Below is a breakdown of the marks for this assignment:

[50 marks] Functionality - Auto-graded Tests

[20 marks] Comments

[10 marks] Code logic and completeness

[10 marks] Code formatting

[10 marks] Meaningful and properly formatted variables

**Total: 100 marks**

We hope these tips help you get started with the Health Information System programming assignment. Remember to test your code using test cases to make sure it is working as expected. Good luck!