# INFORMATION SYSTEMS ASSIGNMENT

README

**EXECUTING THE CODE**

To run the code – ensure Python is installed on your operating system (minimum version: Python 2.7). Ensure that all the python scripts are extracted in the same directory. Open a terminal in this directory and run the following commands:

*>>> python init.py*

The program should now execute and open the main menu for the system. Follow the instructions on-screen and execute the options by entering the numeric values of the user’s choice.

To terminate the program at any stage, press Ctrl + C (Windows/Linux) of Command + . (dot) (iOS).

**CODING APPROACH**

Using an object-oriented approach, each of the given UML class diagrams are related to a class written in python, except for the ini.py scrip, that calls these classes and is responsible for processing the user input to display the relevant options on the CLI. To keep the code easy to understand and read if someone else were to work on it in the future, the relevant classes are split into separate python scripts, each with the class name as title (eg. patient.py representing the Patient class).

As this is an object-oriented system, class methods have been designed to keep information relevant to each instance instantiated. These include class variables to – for example – keep track of which prescriptions are issued to which patient, by which doctor.

The base classes used within the scripts with a short explanation of each are listed below:

* Appointment.py: Containing the Appointment class that creates an instance of an appointment to be added to the relevant doctor’s appointment schedule.
* Appointment\_schedule.py: Containing a pre-populated schedule with some placeholder names as well as timeslots in its appointment\_list variable. This schedule is updated through appointments made through the receptionist.
* Healthcare\_professional: Parent class of Doctor and Nurse.
* Helper.py: A script containing a few helper classes eg. checking input value types.
* Patient.py: Containing Patient class
* Prescription.py: Containing Prescription class that a doctor can issue to a patient.
* Receptionist.py: A class for a receptionist, through whom appointments are made/canceled by a patient instance
* Test.py: An independent script containing functions that were used for testing purposes before implemented in the system
* Init.py: The entry point to the system, this script contains the flow of UI and calls the classes as required by the user.

**LIMITATIONS**

One of the biggest limitations was to not have a persistent database to store class instances in. Each time the application gets terminated, all the data is thus lost and will have to be re-created when restarting the system. For the purpose of demonstration, a couple of doctors, patients, prescriptions and appointments are pre-populated at launch of the init.py script, to have data to work with. Further patients or prescriptions can then be created and stored in the global variables, and can be interacted with as if these were stored in a mySQL database. I tried to implement an instance of mySQL, but limited by the coding platform (codio), importing external libraries/modules are not supported.

Another major limitation was that the python language was limited to python version 2.7. Having gained experience and learning python 3, certain functions could not be handled (eg. using a print statement that does not terminate with a ‘new-line’ (print(“FooBar”,end=’ ’). Working in python 2 was a frustrating experience, and often resulted in hours spent to figure out exceptions thrown, only to learn that this function is deprecated in python 3 or is done in a completely different way (eg. using f-strings to concatenate a print string with variables, or having to revert to raw\_input instead of input for certain user inputs).