**Nombre del autor**:

**Título**: Sistema de atención de cuerpo de guardia de un hospital.

**Número del proyecto**: Proyecto # 20

**Curso**:

**Asignatura**: Programación Orientada a Objetos

**Facultad**: Informática

**Índice**

[Introducción 3](#_Toc115443258)

[Desarrollo 5](#_Toc115443259)

[1.0 Solución del Proyecto 5](#_Toc115443260)

[1.1 Diagrama de Clases 5](#_Toc115443261)

[1.2 Descripción de los métodos usados 8](#_Toc115443262)

[1.3 Descripción de las pruebas unitarias 10](#_Toc115443263)

[Conclusiones 12](#_Toc115443264)

[Recomendaciones 13](#_Toc115443265)

[Bibliografía 14](#_Toc115443266)

# Introducción

Se desea realizar un sistema para el control de los pacientes que llegan a una consulta de cuerpo de guardia de un hospital. Los pacientes que se atienden pueden ser de urgencia o no. De las consultas en general se registra el doctor, la fecha con hora, el paciente, el motivo de la consulta y el código de la consulta. De los pacientes leves se registra además el diagnóstico del médico, si le enviaron análisis y los tratamientos indicados (Higiene-Dietético, Medicina Natural y Tradicional y/o Farmacológico). Por otra parte, de los pacientes de urgencia se registra el síntoma principal, resumen de signos vitales, si lo dejan ingresado o no y el medio de transporte en el que llegó al hospital (Ambulancia, transporte privado o transporte público). De los pacientes se registra el carné de identidad, el nombre, la edad, el sexo, la fecha de nacimiento y la dirección. El resumen de los signos vitales tiene la temperatura, el pulso y las presiones sanguínea mínima y máxima. Del doctor se registra su número de registro profesional, nombre, edad, sexo, fecha de nacimiento, especialidad y si es residente. Por último, se conoce que el sistema cuenta con una lista de consultas, una lista de pacientes y una lista de doctores.

El sistema debe permitir las siguientes funcionalidades:

a) Implemente la funcionalidad necesaria para gestionar (insertar, actualizar, eliminar y listar) los datos de las consultas de forma independiente (urgentes y no urgentes), de los pacientes y de los doctores.

b) Implemente la funcionalidad necesaria para calcular el tiempo promedio que debe durar una consulta dado su código, teniendo en cuenta que se calcula de la siguiente forma:

• Todas las consultas duran al menos 5 minutos para el interrogatorio del paciente.

• Si le envían análisis se demora otros 5 minutos.

• Indicando el tratamiento el médico se demora alrededor de 3 minutos.

• Tomando los signos vitales se emplean 5 minutos más.

• Si el paciente se queda ingresado, los formularios del ingreso toman alrededor de 10 minutos para llenarlos.

c) Implemente la funcionalidad para determinar los signos vitales de un paciente grave que llegó al cuerpo de guardia, dado la fecha y el carné de identidad.

d) Implemente la funcionalidad para determinar qué por ciento de las consultas de urgencia atendieron a pacientes con presión alta teniendo en cuenta que para que sea presión alta la mínima tiene que ser mayor de 90 y/o la máxima mayor de 130.

e) Implemente la funcionalidad para determinar los datos del paciente leve de mayor edad con un diagnóstico dado.

f) Implemente la funcionalidad para determinar el listado de los datos de las consultas de los pacientes leves de un diagnóstico dado que se han consultado en el mes actual, y que esté ordenado cronológicamente por la fecha de la consulta.

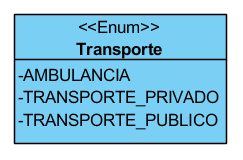
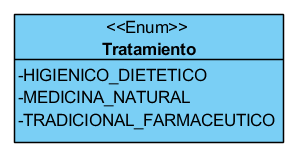
g) Pruebe que las operaciones implementadas en el modelo funcionan correctamente según los datos de prueba que usted le entró al programa.

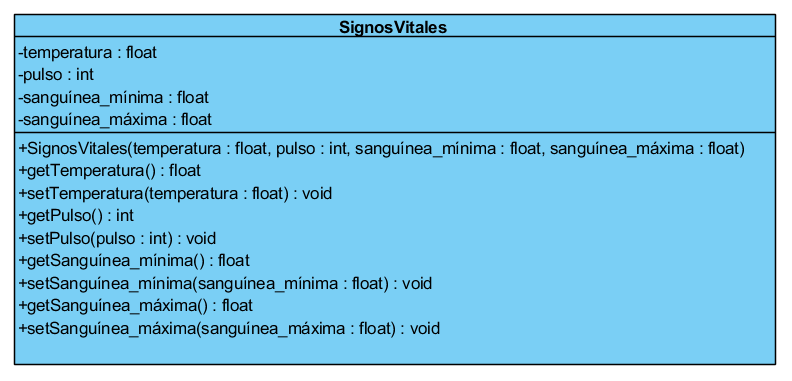
Para desarrollar este proyecto se utilizó PyQt5 es un enlace de la biblioteca gráfica Qt para el lenguaje de programación Python y el editor de código fuente Visual Studio Code para desarrollar el software utilizando Python. (Limited, n.d.)

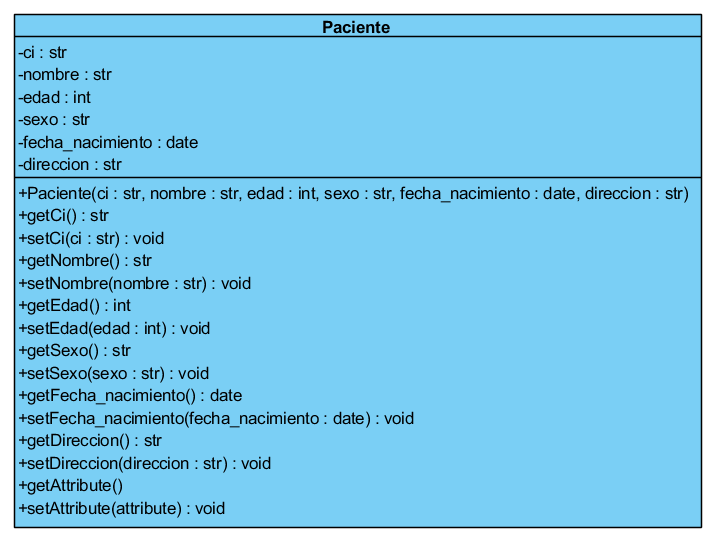
# Desarrollo

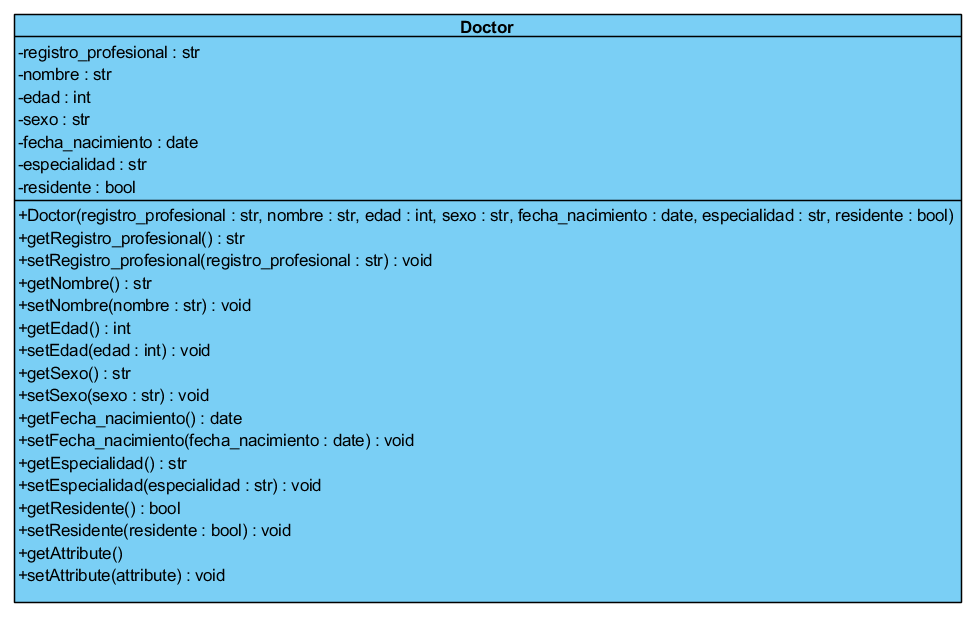
## 1.0 Solución del Proyecto

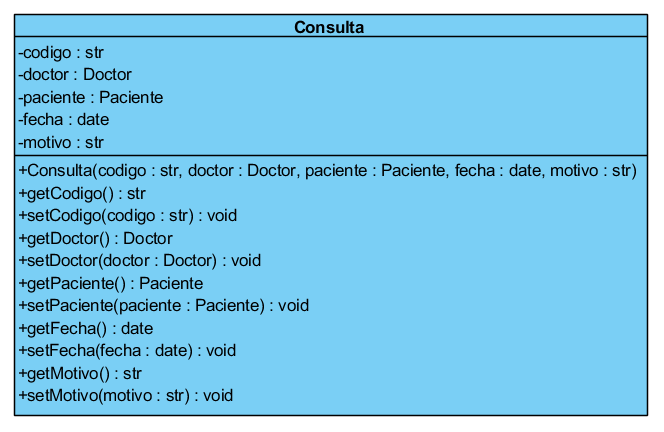
### 1.1 Diagrama de Clases

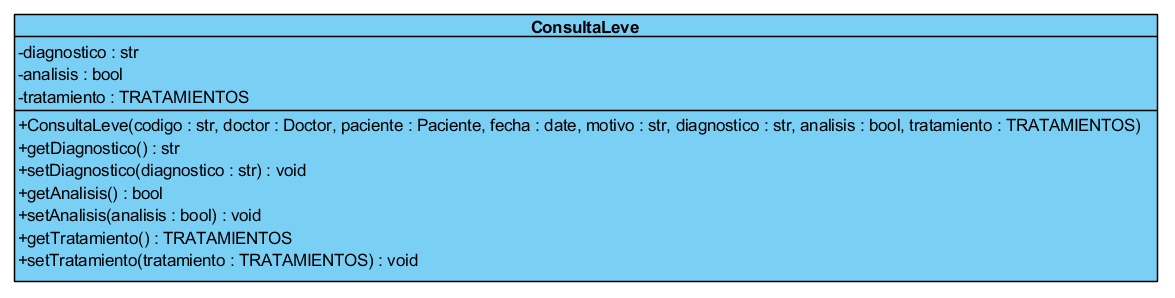
 

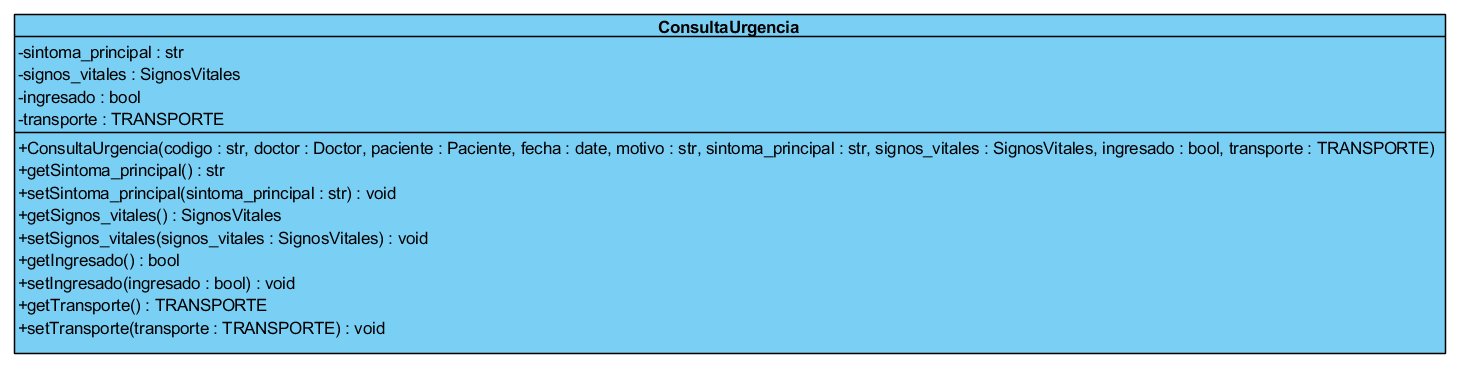


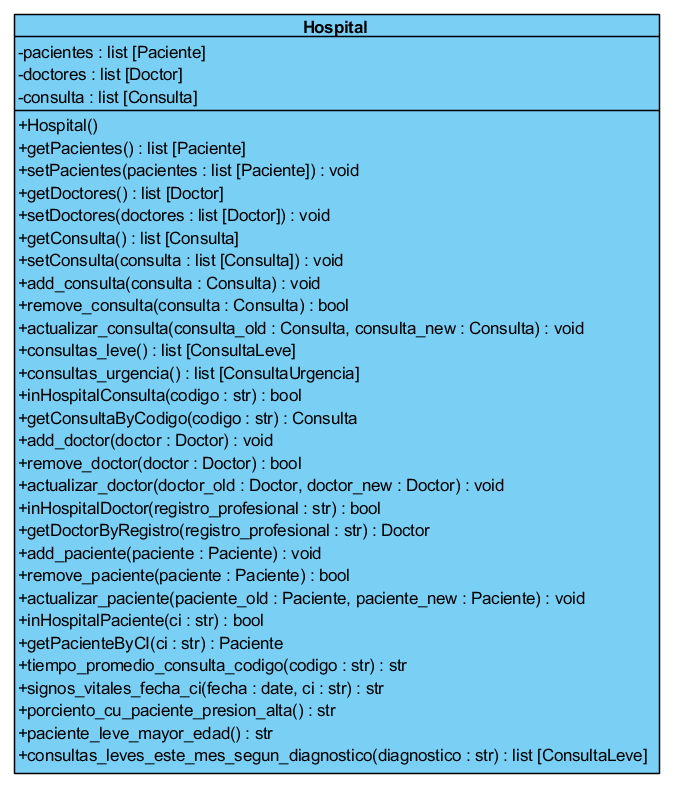


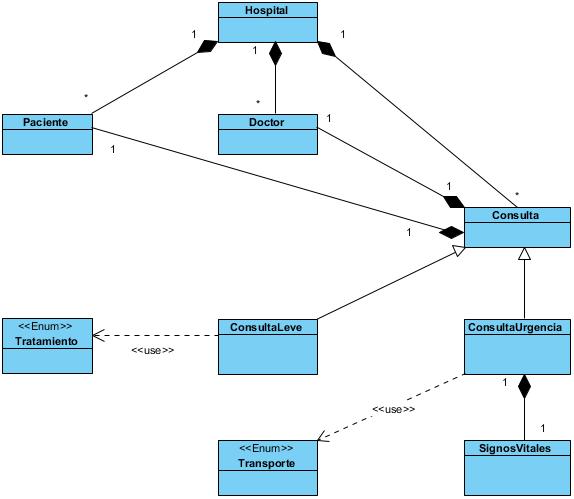












### 1.2 Descripción de los métodos usados

Utilizando los principios de la Programación Orientada a Objeto (learn.microsoft.com, n.d.) en Python según nuestras clases y métodos definidas en el diagrama de clases le dimos solución a las siguientes funcionalidades:

b) Implemente la funcionalidad necesaria para calcular el tiempo promedio que debe durar una consulta dado su código, teniendo en cuenta que se calcula de la siguiente forma:

• Todas las consultas duran al menos 5 minutos para el interrogatorio del paciente.

• Si le envían análisis se demora otros 5 minutos.

• Indicando el tratamiento el médico se demora alrededor de 3 minutos.

• Tomando los signos vitales se emplean 5 minutos más.

• Si el paciente se queda ingresado, los formularios del ingreso toman alrededor de 10 minutos para llenarlos.

Para dar solución a este método era necesario recorrer todas las consultas e ir comparando los códigos hasta encontrar con el que sea igual al **código** pasado por parámetros en el método e inicializar el tiempo\_promedio en 5(descripción del problema todas las consultas se demoran 5min), después de encontrada la consulta preguntamos si es una instancia de **ConsultaLeve** o **ConsultaUrgencia**. Si es una **ConsultaLeve** preguntamos si el análisis es verdadero le agregamos al **tiempo\_promedio** 5 minutos y si además tiene un tratamiento se le suman 3 minutos y el resultado final sería el **tiempo\_promedio** para el caso de **ConsultaLeve**. Si es una **ConsultaUrgencia** preguntamos si la consulta el paciente lleva ingreso, que el atributo ingresado sea verdadero a **tiempo\_promedio** se le suma 10 y si además los signos vitales están definidos que sean todos mayores que cero se le agrega además 5min y tendríamos el valor final en **tiempo\_promedio.** En caso de no encontrar la consulta se retorna en el método que no fue encontrado la consulta por ese código, en caso de que se encontró retornamos un mensaje con el valor del **tiempo\_promedio** calculado.

c) Implemente la funcionalidad para determinar los signos vitales de un paciente grave que llegó al cuerpo de guardia, dado la fecha y el carné de identidad

Para dar solución a esta funcionalidad se recorre el listado de consultas se compara que la **fecha** entrada por parámetros sea igual a la **fecha** de la consulta recorrida, que se cumpla también que el **carnet de identidad** del paciente entrado por parámetros sea igual al **carnet de identidad** del paciente de la consulta en cuestión y además que la consulta sea una instancia de **ConsultaUrgencia**, si se cumplen estas condiciones entonces retornamos un mensaje con los **signos vitales** de la consulta encontrada, si no se retorna un mensaje informando que no se encontró una consulta con la **fecha** y el **carnet de identidad** entrado por parámetros.

d) Implemente la funcionalidad para determinar qué por ciento de las consultas de urgencia atendieron a pacientes con presión alta teniendo en cuenta que para que sea presión alta la mínima tiene que ser mayor de 90 y/o la máxima mayor de 130.

Para dar solución a esta funcionalidad se recorre las consultas de la propiedad **consultas\_urgencias,** se comparan los signos vitales de la consulta en cuestión, que la sanguínea mínima se mayor que 90 y la máxima con 130, si se cumple incrementamos en 1 el valor de la variable temporal **paciente\_presion\_alta** inicializada en 0, la cual al final de recorridas todas las consultas va a tener el valor de todas las que cumplen la condición de presión alta, se divide entre la cantidad de elementos de la lista de la propiedad **consultas\_urgencias** y se devuelve un mensaje con este valor calculado.

e) Implemente la funcionalidad para determinar los datos del paciente leve de mayor edad con un diagnóstico dado.

Se recorren la lista de la propiedad **consulta\_leve**, si se cumple la condición que el diagnostico no este vacío, se guarda el paciente en la variable **paciente\_mayor** si su valor es None, si su valor no es **None** se compara con el paciente de la consulta que estamos iterando y si su edad es mayor entonces cambia su valor por el paciente de la consulta en cuestión. Al final de recorrida la lista se obtiene el paciente de mayor edad en la variable **paciente\_mayor**, de ser su valor None retornamos un mensaje que no había consultas leves en el hospital en caso contrario retornamos un mensaje con sus datos.

f) Implemente la funcionalidad para determinar el listado de los datos de las consultas de los pacientes leves de un diagnóstico dado que se han consultado en el mes actual, y que esté ordenado cronológicamente por la fecha de la consulta.

Se recorre la lista de la propiedad **consulta\_leve** y se pregunta si el diagnóstico de la consulta es igual al entrado por parámetros y que además el año y el mes de la fecha de la consulta sea igual al de la fecha actual entonces se agrega a una lista para retornarla. Después de llenada la lista se ordena según el día de la fecha de la consulta y se retorna la lista ordenada.

### 1.3 Descripción de las pruebas unitarias

Para el correcto funcionamiento de nuestro software se crearon las pruebas unitarias utilizando del paquete unittest de Python la clase TestCase. (docs.python.org, n.d.)

Para la realización de la prueba en la clase Paciente de nuestro software se crearon varias pruebas como son el test de getters, setters de todos los atributos y el constructor donde todos los datos de prueba fueron satisfactorios. Fue necesario para que nuestro código funcionaria a cualquier dato entrado por el usuario manejar las excepciones de entrada de los datos de nuestros atributos, validando así en la clase Paciente que el carnet de identidad fuera un string con 11 caracteres numéricos, en caso de no ser así lanza una excepción, en el nombre que fuera un string con no menos de 3 caracteres y no mas de 100, la edad de tipo int no menor de 0 y no mayor de 150, el sexo obligatorio fuera Masculino o Femenino de tipo string, la fecha de tipo date, no menor a la fecha actual restándole 150 y no mayor que la fecha actual, la dirección que fuera un string no menor de 10 caracteres y no mayor de 254 caracteres. Utilizando los datos de pruebas todos los test resultaron positivos y arrojaron los datos esperados.

# Conclusiones

Con la realización de este proyecto profundizamos los conocimientos de la programación orientada a objetos, aprendimos a diseñar vistas gráficas utilizando la librería PyQt5 con el lenguaje de programación Python, vinculamos nuestras clases modelo con las vistas de diseño, manejamos los posibles errores que el usuario pudiera cometer a la hora de introducir datos en tiempo de ejecución de tal manera que nuestro programa no colapse. Generamos un ejecutable usando la librería cx-Freeze==6.11.1 (cx-freeze.readthedocs.io, n.d.) logrando así crear un software para el control de las consultas de los pacientes en un hospital.

# Recomendaciones

Recomendamos para poner en funcionamiento en Linux generar el ejecutable para el sistema operativo Linux, dado que ya generamos el ejecutable para Windows funcionando perfectamente después de pasado las pruebas a el software.

Para futuras versiones agregar nuevas funcionalidades que ayuden a los doctores del hospital el registro de todas las consultas de un paciente para llevar un historial clínico de los mismos logrando así un mejor diagnostico y agilizando que tratamientos debe llevar el paciente.

# Bibliografía

cx-freeze.readthedocs.io. (s.f.). Obtenido de cx-freeze.readthedocs.io: https://cx-freeze.readthedocs.io/en/latest/index.html

docs.python.org. (s.f.). Obtenido de docs.python.org: https://docs.python.org/es/3/library/test.html

learn.microsoft.com. (s.f.). Obtenido de learn.microsoft.com: https://learn.microsoft.com/es-es/training/modules/python-object-oriented-programming/

Limited, R. C. (n.d.). riverbankcomputing.com. Retrieved from riverbankcomputing.com: https://www.riverbankcomputing.com/static/Docs/PyQt5/