

**UNIVERSIDAD NACIONAL DE TRUJILLO**

Facultad de Ingeniería

Escuela Profesional de Ingeniería Mecatrónica

DESARROLLO DE UNA APLICACIÓN PARA LA GESTIÓN DE HISTORIALES MÉDICOS Y DE UN OXIMETRO DE PULSO PORTATIL



**PROYECTO DE PROGRAMACION II**

**AUTORES:**

**Ortiz Salvador, Edinson Elias**

**Sánchez Rojas, Jhonatan Artemio**

**Valdez Julca, Luis Alberto de Jesús**

**Valdiviezo Jiménez, Víctor Javier**

**Vigo Villar, Cristhian Aaron**

**DOCENTE :**

**Escobedo Cárdenas, Edwin Jonathan**

**Roca Reyes, José Francisco**

**CICLO :**

**2022 - II**

Trujillo, Perú 2022

# Resumen

El presente informe consiste en la elaboración de un proyecto de programación que busca la implementación de historiales clínicos digitales en aras de la agilización en el servicio médico, mediante la creación de un programa de escritorio, el cuál contribuya al almacenamiento y consultoría de historiales clínicos, donde el medico tendría acceso rápido a la ‘hoja de vida’, referida a asuntos netamente clínicos, del paciente, como atenciones, diagnósticos, prescripciones, tratamientos, datos, entre otros. Además, de su nivel de saturación de oxígeno, dado por un oximetro de pulso de fabricación propia. El desarrollo del software requerirá del uso de herramientas de programación como Python, visual studio code, biblioteca gráfica Tkinter y el sistema de gestión para bases de datos MySQL Workbench, mientras que el desarrollo del hardware requerirá el uso de un Arduino. La motivación que sigue el proyecto es la busca de la mejora progresiva en la atención médico-paciente, evitando así, el desgaste de los procesos internos administrativos en los centros médicos como el uso de hojas de papel y archivos físicos.

***Palabras Claves: Python, Tkinter, MySQL, historial médico***

# Abstract

This report consists of the development of a programming project that seeks the implementation of digital medical records for the sake of speeding up the medical service, through the creation of a desktop program, which already contributes to the storage and consulting of medical records, where the doctor has quick access to the 'resume', referring to purely clinical matters, of the patient, such as care, diagnoses, prescriptions, treatments, data, among others. In addition, their oxygen saturation level, given by a self-made pulse oximeter. Software development requires the use of programming tools such as Python, visual studio code, the Tkinter graphics library, and the MySQL Workbench database management system, while hardware development requires the use of an Arduino. The motivation behind the project is the search for progressive improvement in doctor-patient care, thus avoiding the wear and tear of internal administrative processes in medical centers such as the use of sheets of paper and physical files.

***Keywords: Python, Tkinter, MySQL, medical historial***

**Tabla de Contenidos**

[Resumen 2](#_Toc120908815)

[Abstract 3](#_Toc120908816)

[Capítulo 1 Introducción 5](#_Toc120908817)

[1.1. Formulación del Problema 5](#_Toc120908818)

[1.2. Objetivos 5](#_Toc120908819)

[1.3. Justificación 6](#_Toc120908820)

[Capítulo 2 Marco Teórico 7](#_Toc120908821)

[Capítulo 3 Procedimientos 9](#_Toc120908822)

[3.1. Descripción general de los procedimientos 9](#_Toc120908823)

[3.2. Desarrollo de los procedimientos 11](#_Toc120908824)

[3.2.1. Implementación de la Base de Datos: 11](#_Toc120908825)

[3.2.2. Elaboración del código ejecutor del programa y de la interfaz principal 18](#_Toc120908826)

[3.2.3. Implementación de página web 34](#_Toc120908827)

[3.2.4. Elaboración del Oximetro de pulso portátil 34](#_Toc120908828)

[Capítulo 4 Resultados 35](#_Toc120908829)

[4.1. Aplicación de escritorio 35](#_Toc120908830)

[4.2. Página web 35](#_Toc120908831)

[4.3. Oximetro de pulso portátil 35](#_Toc120908832)

[Capítulo 5 Conclusiones y Recomendaciones 35](#_Toc120908833)

[5.1. Conclusiones 35](#_Toc120908834)

[5.2. Recomendaciones 37](#_Toc120908835)

[Referencias Bibliográficas 40](#_Toc120908836)

[ANEXOS 44](#_Toc120908837)

# Capítulo 1 Introducción

Aun estando en un siglo donde la tecnología es lo que se prioriza, siguen existiendo centros médicos que tramitan las historias clínicas manualmente, en otras palabras, las tienen almacenadas de manera física. Y en la mayoría de circunstancias, este hecho provoca un gran retraso en el tiempo de servicio que se le da a los pacientes que llegan para ser atendidos en dicho centro. El hecho de hacer la búsqueda de una historia clínica en un almacén y luego llevarlo hasta el consultorio médico, aumenta notoriamente el tiempo de atención al paciente. Este proyecto busca dar una solución a este problema, al diseñar un sistema de control para historiales médicos, el cual permita una gestión más organizada y automatizada de las historias clínicas en un centro de atención médica, para perfeccionar el servicio de atención a sus pacientes.

Además, sin ser ajenos a la situación mundial pos pandemia de la cual todos estamos siendo partes, también se creyó conveniente la realización de un oximetro de pulso que ayude a controlar los niveles de saturación de oxígeno en la sangre y poder almacenar estos datos en la misma aplicación.

## Formulación del Problema

¿Cómo desarrollar una aplicación para la gestión de historiales médicos y niveles de saturación de oxigeno que mejore la calidad de servicio de un centro médico?

## Objetivos

* + 1. **Objetivo General**

Desarrollar una aplicación para la gestión de historiales médicos y niveles de saturación de oxígeno de los pacientes.

* + 1. **Objetivos Específicos**
       1. Implementar la base de datos en MySQL workbench.
       2. Elaborar el código ejecutor del programa y de la interfaz principal.
       3. Implementar una página web.
       4. Fabricar un oxímetro de pulso portátil.

## Justificación

* **Social:** Mejorar la calidad de servicio y atención a los pacientes.
* **Económica:** Disminuirá la inversión en los materiales de oficina, en este caso, el uso del papel bond.
* **Tecnología:** Generar un antecedente para mejoras futuras en la implementación de estas interfaces.

# Capítulo 2 Marco Teórico

**Historia clínica**

Es el documento médico legal donde queda registrada toda la relación del personal sanitario con el paciente, todos los actos y actividades médico-sanitarias realizados con él y todos los datos relativos a su salud.

Tiene como finalidad primordial recoger datos del estado de salud del paciente con el objeto de facilitar la asistencia sanitaria. El motivo que conduce al médico a iniciar la elaboración de la historia clínica y a continuarla a lo largo del tiempo, es el requerimiento de una prestación de servicios sanitarios por parte del paciente.

**Oximetro de pulso**

Es un dispositivo que mide la saturación de la sangre sin necesidad de una prueba de laboratorio, son portátiles y pequeños.

Lo ideal es que los niveles de oxígeno en la sangre se encuentren entre 95% y 98%. Con el oxímetro se puede determinar una disminución anormal de oxígeno en la sangre o más conocida como hipoxemia, cuando se obtiene una saturación de oxígeno menor a 90%.

Uno de los síntomas relacionados al COVID-19 es la dificultad para respirar, en algunos casos esto no es percibido en su etapa inicial. Al no ser detectado a tiempo puede dar lugar a un cuadro más grave llamado hipoxia que deteriora las funciones corporales de la persona.

Por este motivo es importante monitorear los niveles de oxígeno en la sangre para detectar los síntomas en la etapa inicial de la enfermedad y afrontar un mejor proceso de recuperación.

**MAX 30102**

Es un sensor que incorpora las funciones de pulsímetro y oxímetro en un único integrado que podemos usar junto con un procesador como Arduino.

La serie MAX3010x es un sensor óptico, que basa su funcionamiento en el distinto comportamiento que la sangre tiene ante la luz, en función de su grado de saturación de oxígeno. La sangre oxigenada absorbe mayor cantidad de luz infrarroja, mientras que la sangre poco oxigenada absorbe mayor luz roja.

Para ello, el MAX30102 incorpora dos LED, uno de espectro rojo y otro de infrarrojo. El MAX30102 se pone sobre la piel, por ejemplo, en el dedo o la muñeca. El sensor detecta la luz reflejada, y determina el grado de saturación.

# Capítulo 3 Procedimientos

## Descripción general de los procedimientos

* + **Aplicación de escritorio**

Para el desarrollo de la aplicación se hizo necesario el uso de MySQL workbench, Python y Tkinter, los cuales se juntan para realizar la parte de base de datos, código e interfaz, respectivamente.

Inicialmente se crean las tablas necesarias en el MySQL. Luego, usando el lenguaje Python, se crea la conexión con estas tablas. Finalmente, se genera el código de las interfaces, con la ayuda de la librería Tkinter.

* + **Página web**

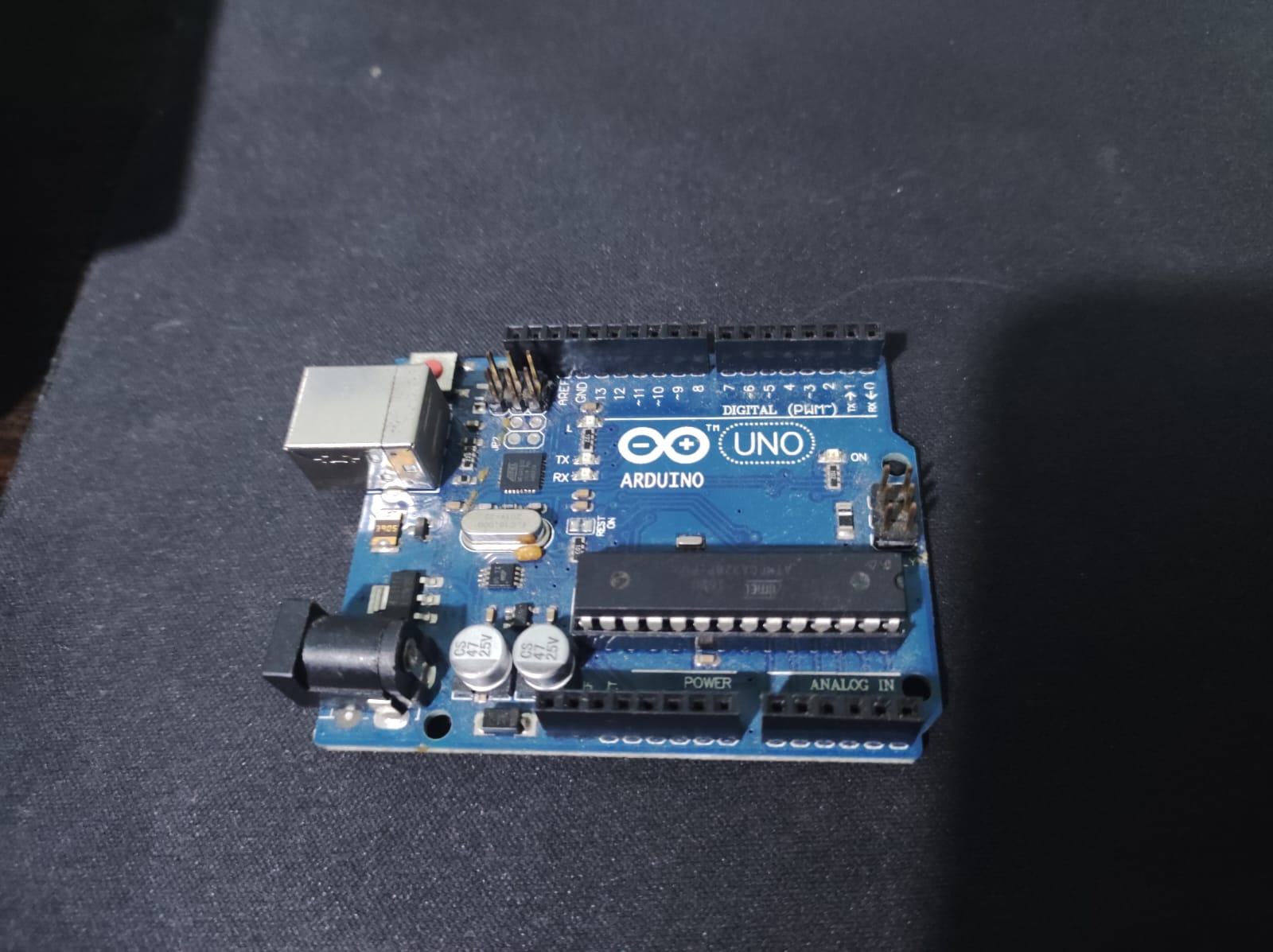
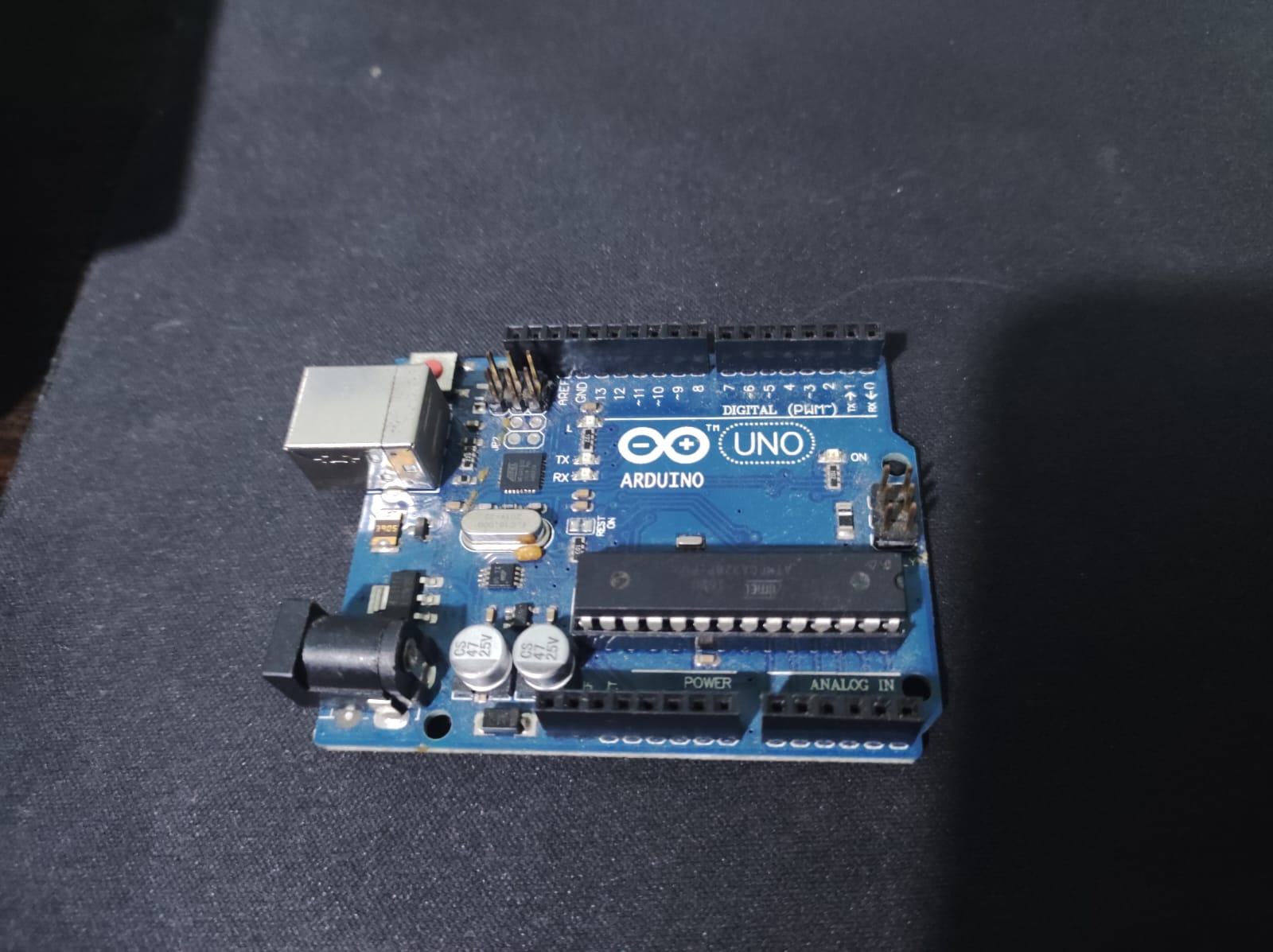
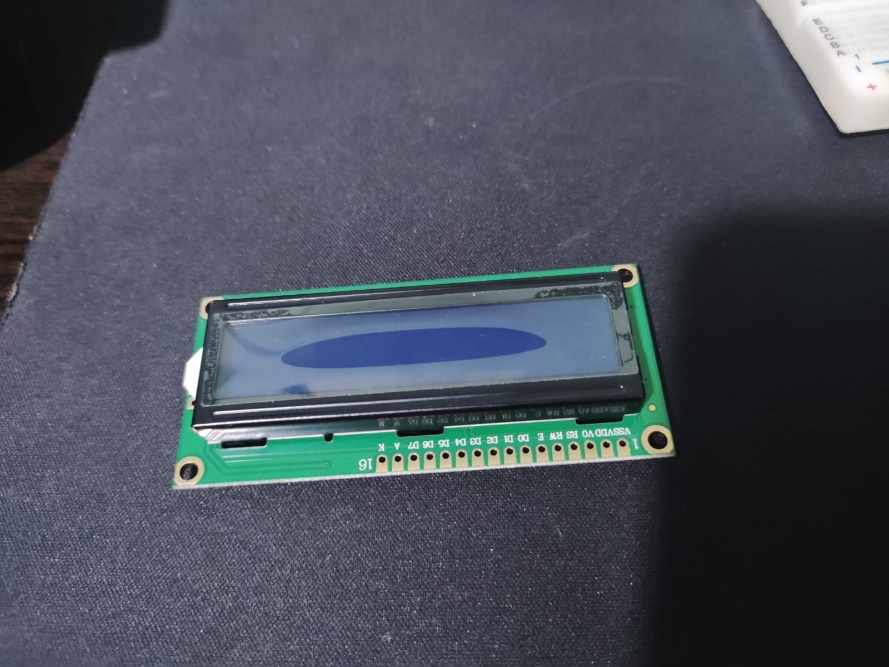
Usando los pseudolenguajes de programación HTML y CSS, se genera una página web para la presentación del centro médico al que pertenece la aplicación de escritorio. El contenido de esta página se basará en información general del centro médico en mención, como su ubicación, horario de atención, especialidades, entre otros. Por lo que se considera un complemento del beneficiario directo de la aplicación, mas no de la aplicación como tal. De esta manera, la página web tendrá acceso abierto para público en general como también, un apartado en la aplicación para su apertura desde la misma.

* + **Oximetro de pulso Portátil**

Utilizando el lenguaje de programación Arduino, y haciendo uso de un módulo MAX 30102 junto a un Display, se creará un oximetro portátil el cual será usado por el personal del hospital, logrando así, medir saturación de oxígeno o los niveles de oxígeno en la sangre a las personas. La creación del oximetro digital se llevará acabo uniendo a nuestro Arduino el modulo MAX 30102 y el Display,

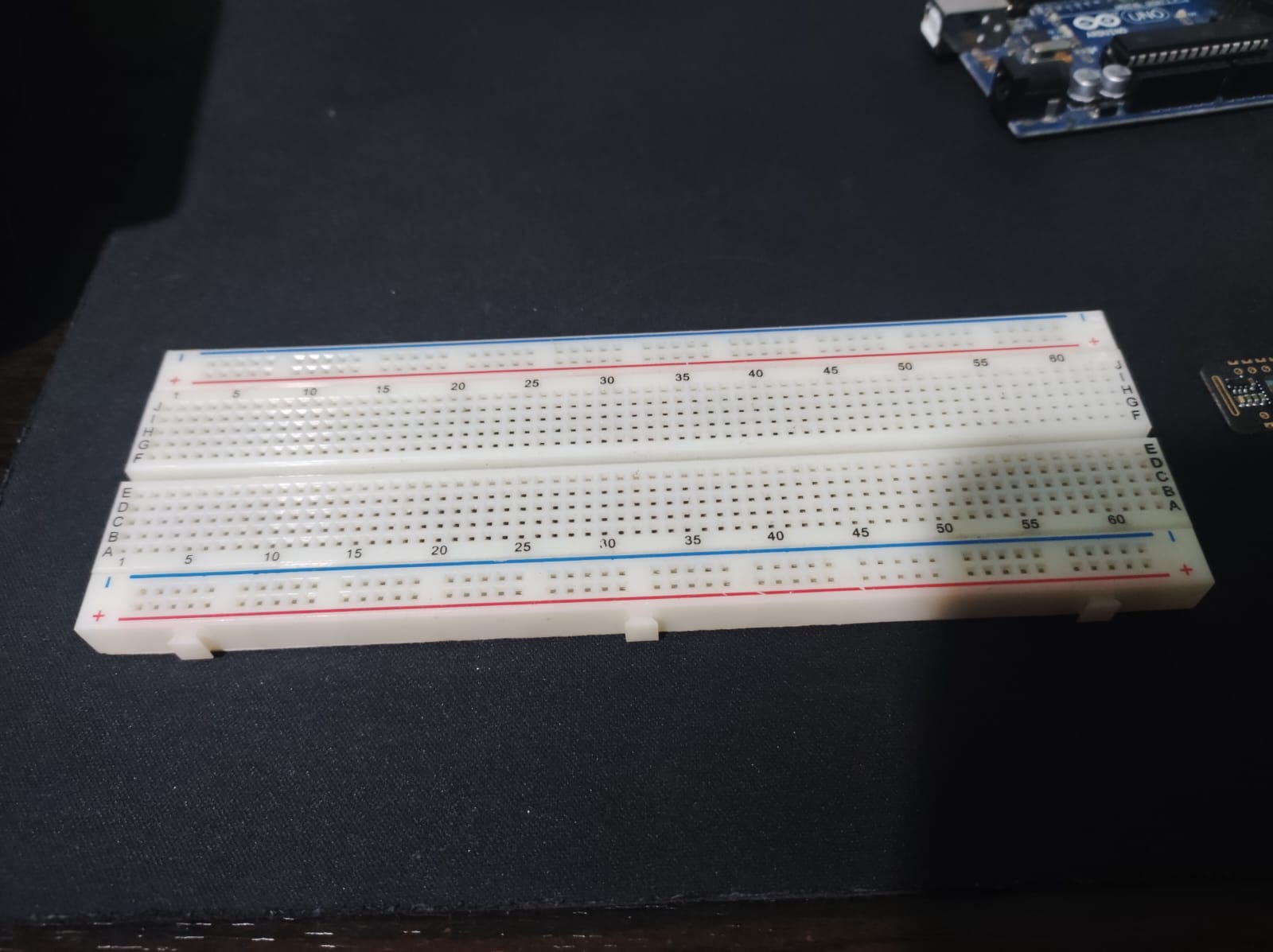
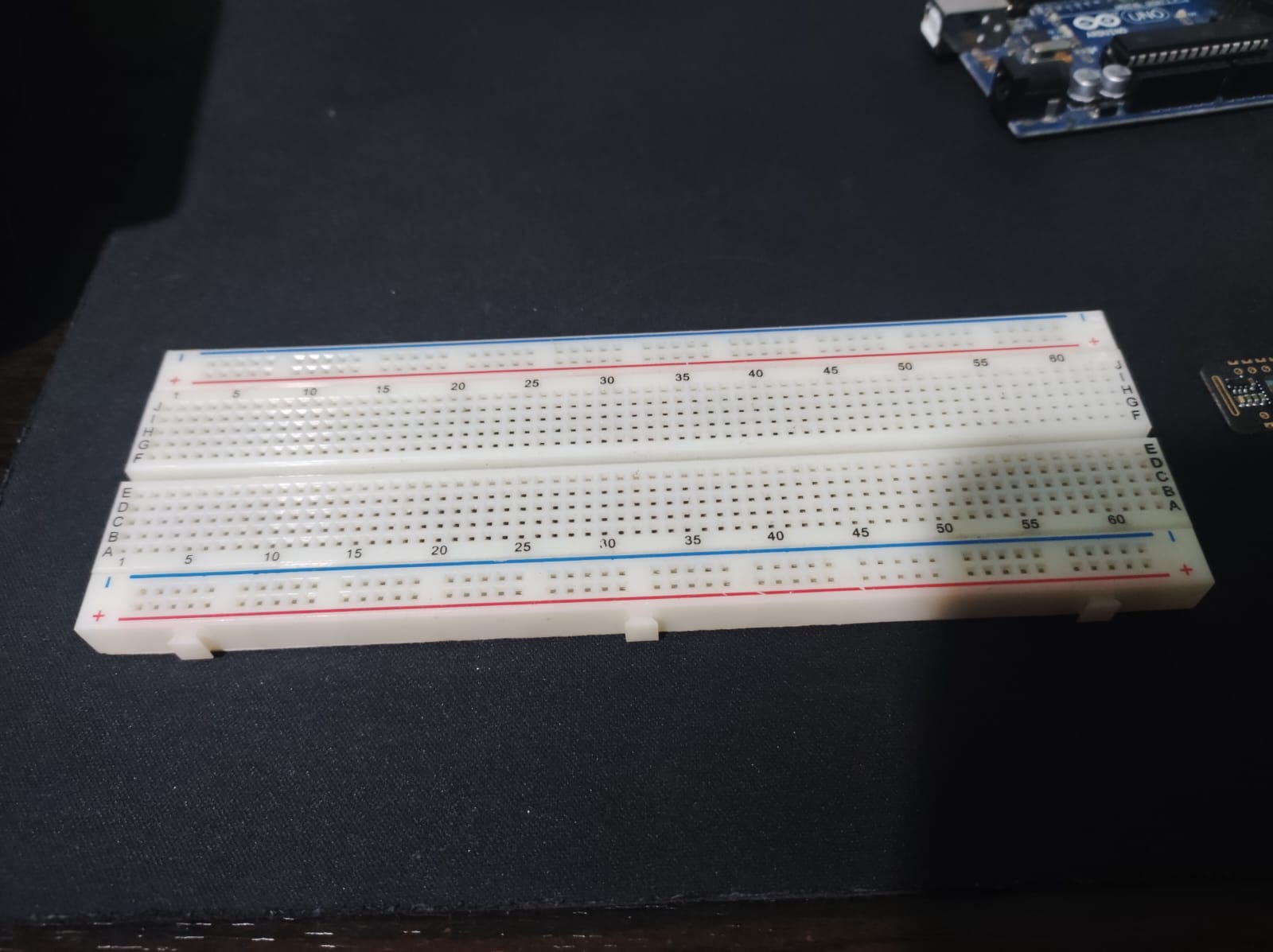
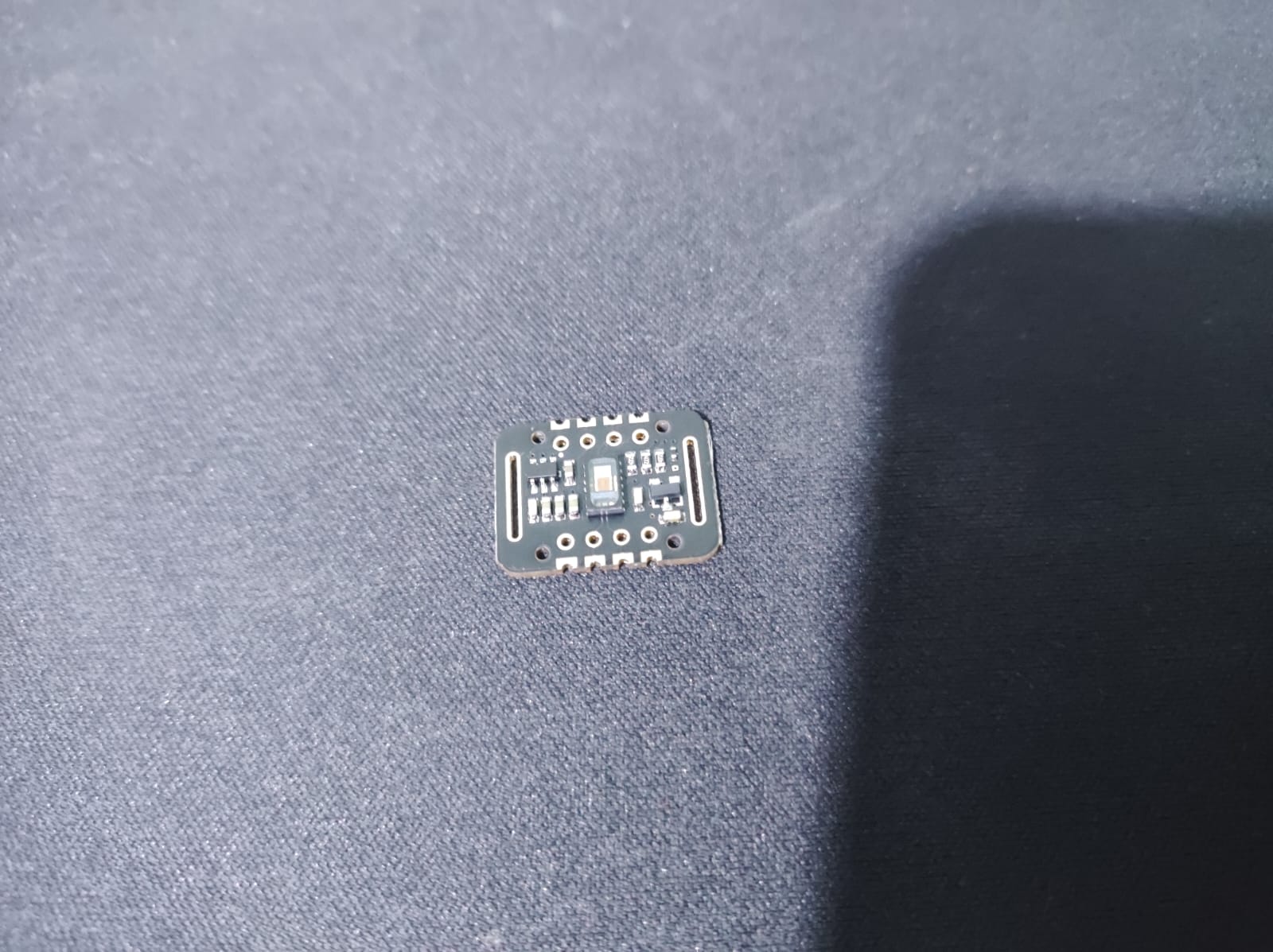
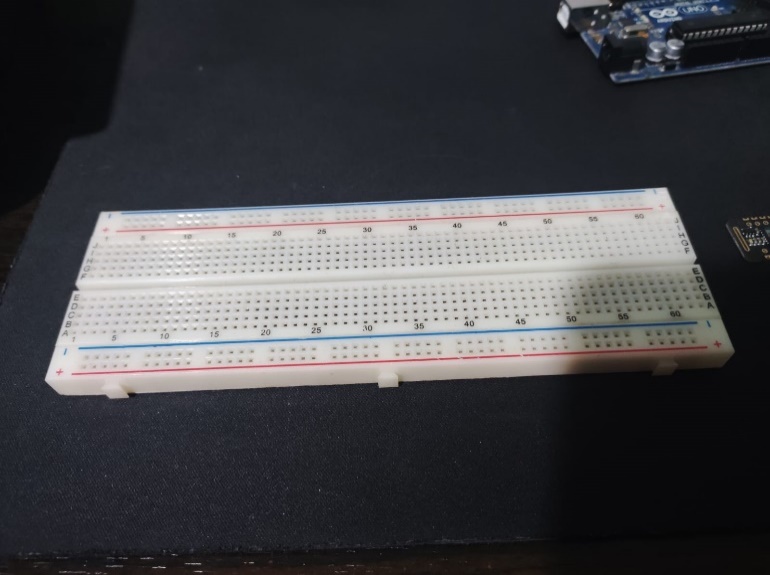
mediante el uso del protoboard y jumpers para facilitar las conexiones, y de ser necesaria soldar algunas piezas para un mejor desempeño. Una vez realizada las conexiones, se buscará crear un código en Arduino, el cual ayude a manejar los datos recibidos del módulo y mostrarlos de manera puntual en Display.

Algunos componentes importantes que se utilizaran:



Arduino

Display



Protoboard

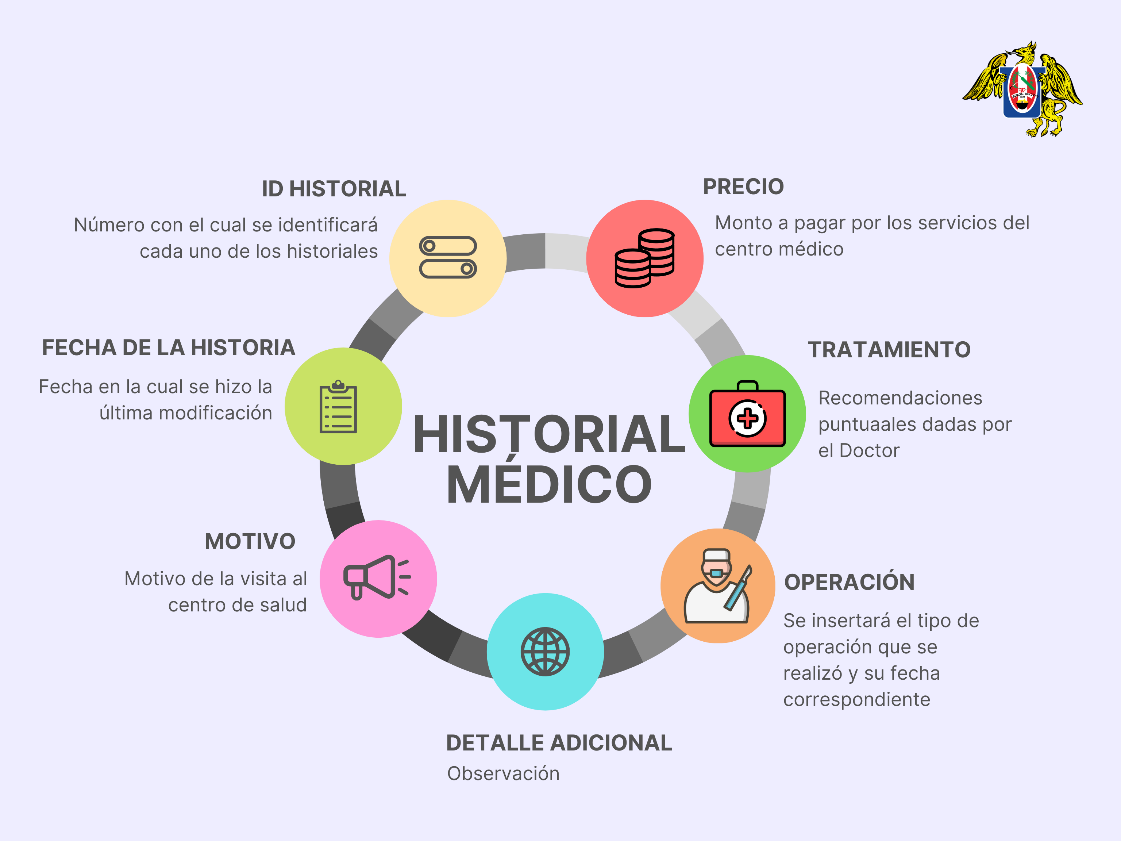
MAX 30102

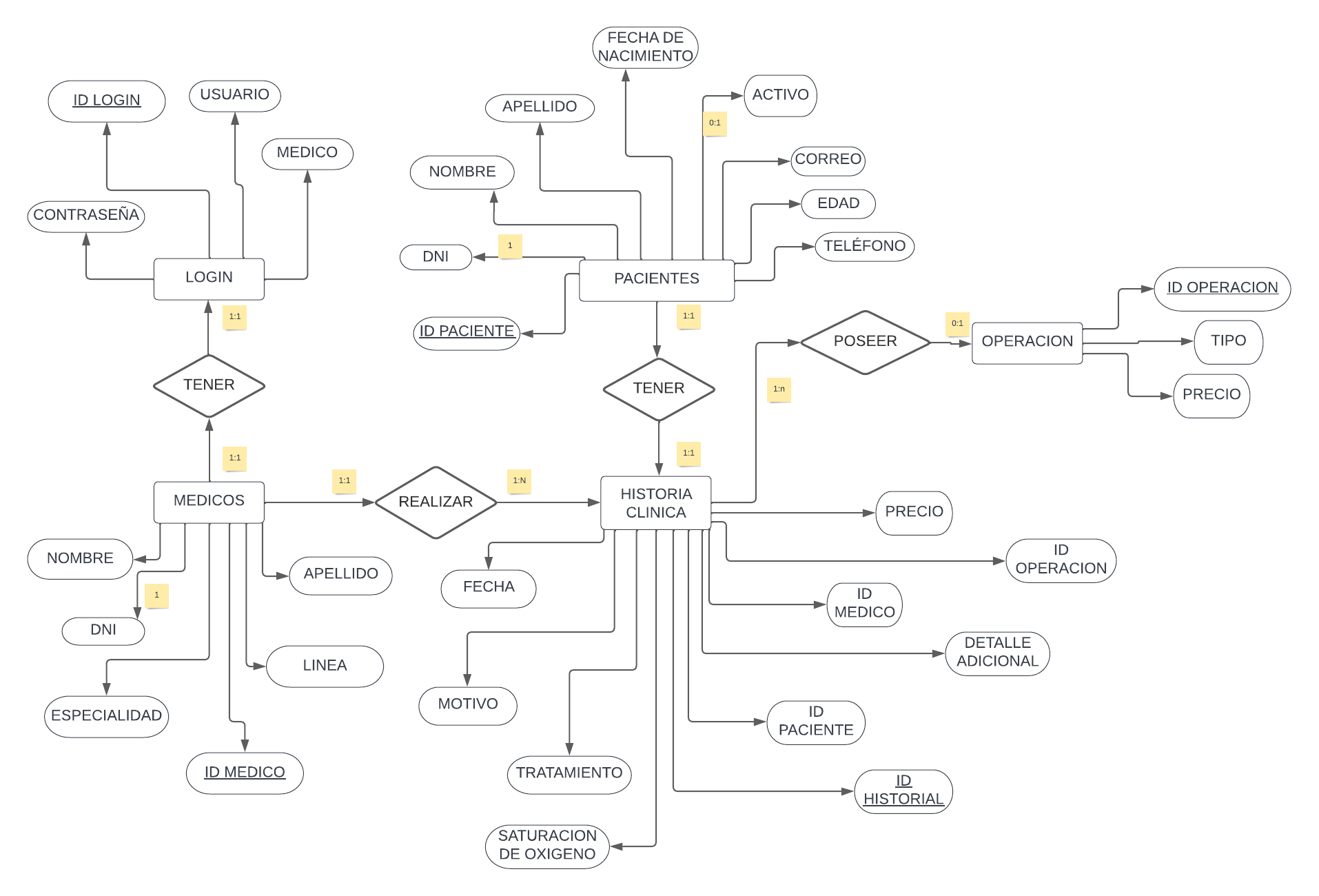
## Desarrollo de los procedimientos

## Implementación de la Base de Datos:

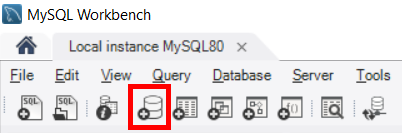
Para la implementación de la base de datos, se usó el software MySQL Workbench, en dicho software creamos las tablas que posteriormente se importaran al código general para hacer la debida conexión con este mismo.

* Iniciamos identificando las entidades principales y los datos que se quieren guardar de estas mismas, para lo cual nos apoyamos de los siguientes gráficos:

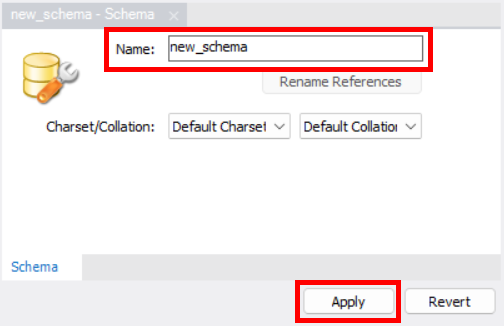


* Con estos datos generales identificados, pasamos a la realización del diagrama E-R. Tomando en cuenta futuras entidades que se quieran relacionar pero que no están presentes en los anteriores gráficos.
* Iniciamos creando la tabla que contendrá los datos del paciente y de su historial. El software MySQL Workbench posee su propio lenguaje de programación, a continuación, se explicará paso a paso:

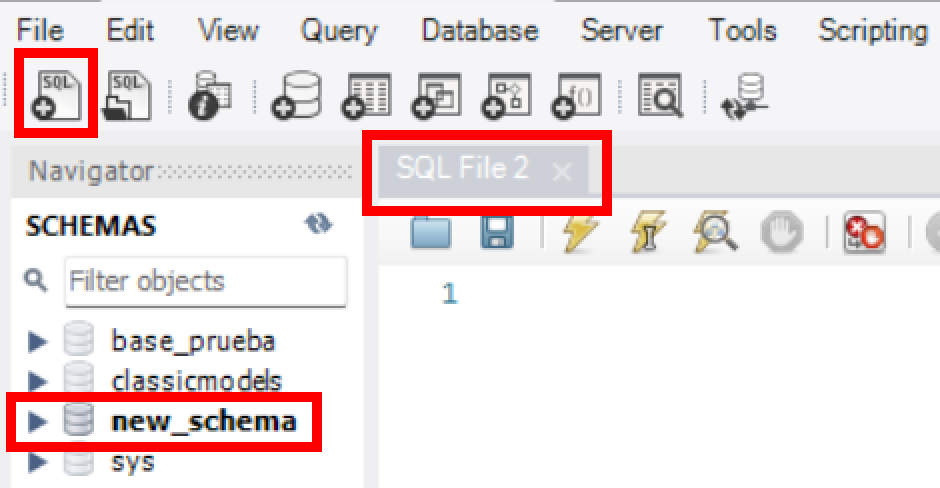
1. Abrimos la aplicación y damos click en el siguiente icono para crear nuestra base de datos.



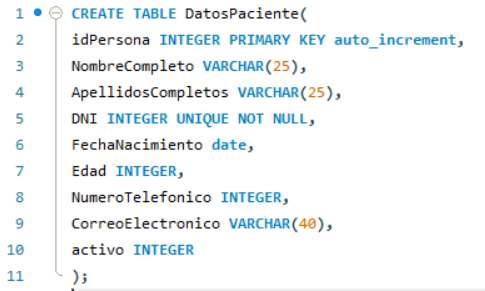
1. Se abre la siguiente sección en el cual colocamos el nombre de la base de datos y luego le damos click en “Apply”, luego se abrirá una ventana emergente al cual también le damos click en “Apply”.



1. En la parte izquierda de la interfaz veremos la base de datos que hemos creado, damos doble click sobre nuestra base para activarlo y luego damos click en el icono de la parte superior izquierda y nos aparecerá una ventana para comandos.



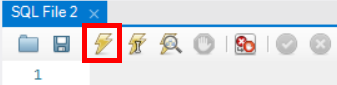
1. Se digitó el siguiente código en la ventana de comandos para crear la tabla que contenga los datos del paciente:

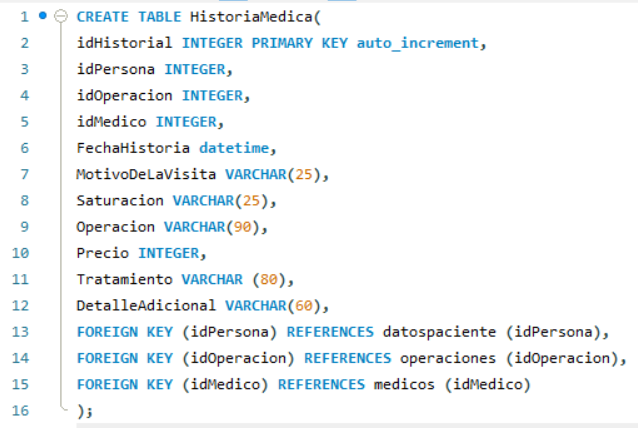


Donde:

* CREATE TABLE: Este comando nos permite crear la tabla.
* INTEGER: Define un dato como entero.
* INTEGER UNIQUE: Similar al comando INTEGER, solo que al tener el UNIQUE, garantiza que no se duplique en las columnas o secciones establecidas.
* VARCHAR: Define un dato como cadena o texto.
* PRIMARY KEY: Define un dato como único y además de tomar valores que no sean nulos, para poder identificar de forma exclusiva cada fila de las tablas.
* AUTOINCREMENT: Este comando es para generar un número único cuando se inserta algún nuevo registro en la tabla.

1. Posteriormente al haber colocado el código, compilaremos el código para que se ejecute y cree la tabla si todo está correcto:

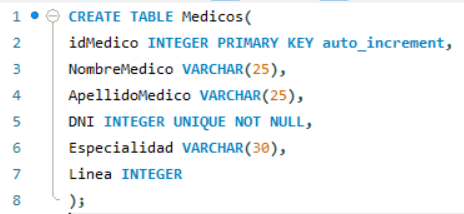


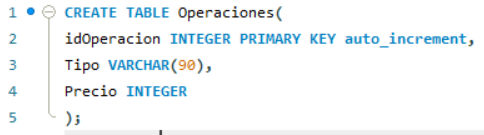
1. De manera análoga se repitió los pasos anteriores para crear la tabla de Historia Médica:

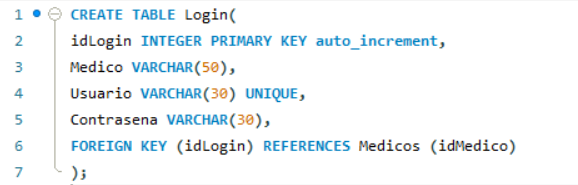
Dos nuevos comandos aparecen en este caso:

* FOREIGN KEY: Sirve para señalar la clave primaria (PRIMARY KEY) de otra tabla.
* REFERENCES: Esto acompaña a FOREIGN KEY para señalar a que tabla está haciendo referencia.

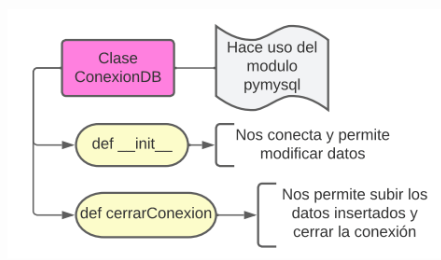
1. Similar proceso para crear las tablas Médicos, Operaciones y Login:







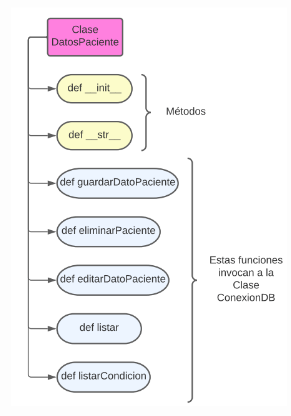
## Elaboración del código ejecutor del programa y de la interfaz principal

**Conexión.py:**

Esta clase consiste de dos métodos:

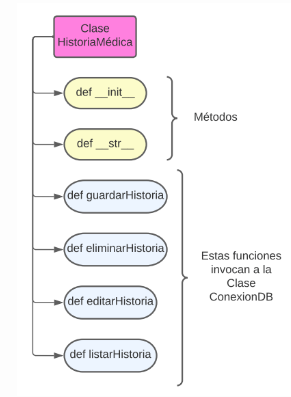
* **def \_\_init\_\_:** Nos permitirá conectarnos a la base de datos mediante el módulo ‘pymysql’, dicha conexión contiene el host, user, password y db. Para modificar los datos se hará uso del método ‘cursor’.
* **def cerrarConexion:** Para poder subir los datos insertados en las entries se hace uso del método ‘commit’ y para cerrar la conexión luego de haber insertado los datos, se hará uso del método ‘close’.

Esta clase es fundamental, dado que, las demás clases la van a invocar al momento de ejecutarse.

**PacienteDao.py:**

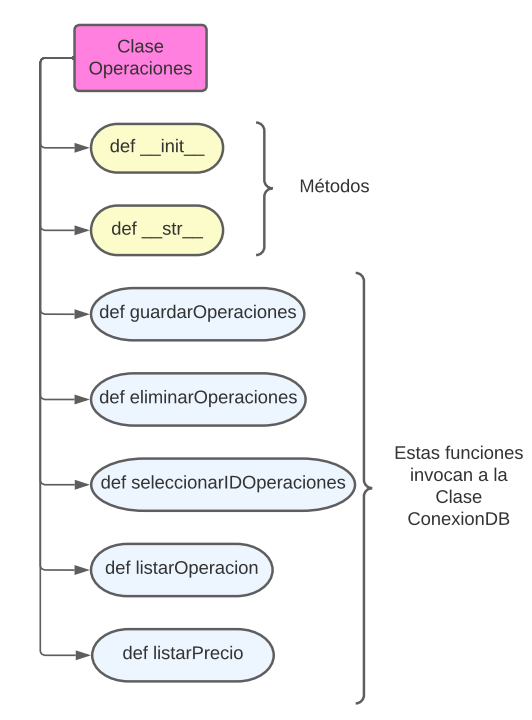
Esta clase consiste de dos métodos y cinco funciones:

* **Métodos:**
* **def \_\_init\_\_:** Es un método constructor, cuyos parámetros son los nombres de las columnas de la tabla DatosPaciente (NombreCompleto, ApellidosCompletos, DNI, FechaNacimiento, Edad, NumeroTelefonico, CorreoElectronico)
* **def \_\_str\_\_:** Este método muestra los objetos del método anterior (self.NombreCompleto, self.ApellidosCompletos, self.DNI, etc)
* **Funciones:**
* **def guardarDatoPaciente:** Esta función nos permitirá guardar los datos del paciente, además, el parámetro para esta función será “persona”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia a cada objeto persona de la tabla DatosPaciente (persona.NombreCompleto, persona,ApellidosCompletos, persona.DNI, etc). Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox).
* **def eliminarPaciente**: Esta función vuelve el activo de un paciente a 0, no será borrado de la base de datos, pero no podrá ser visualizado en la tabla de la GUI. El parámetro para esta función será “idPersona”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia al parámetro “idPersona” y así poder volver su activo a 0. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox), el mensaje de la ventana emergente puede variar dependiendo si se eliminó exitosamente o si hubo algún error.
* **def editarDatoPaciente:** Esta función es para sobrescribir en cada objeto persona de la clase DatosPaciente. Los parámetros para esta función serán “idPersona” y “persona”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia al parámetro “idPersona” y a los objeticos persona para así poder editar su contenido. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox), el mensaje de la ventana emergente puede variar dependiendo si se editó exitosamente o si hubo algún error.
* **def listar:** Esta función manda los datos a la lista ListaDatosPaciente (Tabla en la GUI). Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, una variable hará referencia a la lista vacia en donde se guardarán los datos para mostrarlos (ListaDatosPaciente) y otra variable (sql) la cual referencia que se muestren los pacientes activos. Posteriormente se ejecutará la variable “sql”, se añadirá los datos a la lista y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. Todo retorna a la ListaDatosPaciente.
* **def listarCondicion:** Esta función manda los datos a la lista ListaDatosPaciente (Tabla en la GUI) si cumplen con la condicion where. El parámetro de esta función será “where”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, una variable hará referencia a la lista en donde se guardarán los datos para mostrarlos (ListaDatosPaciente) y otra variable (sql) la cual referencia a la condición where (solo los pacientes activos). Posteriormente se ejecutará la variable “sql”, se añadirá los datos a la lista y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. Todo retorna a la ListaDatosPaciente. Esta función es importante para así poder visualizar solo a los pacientes activos.

**HistorialDao.py:**

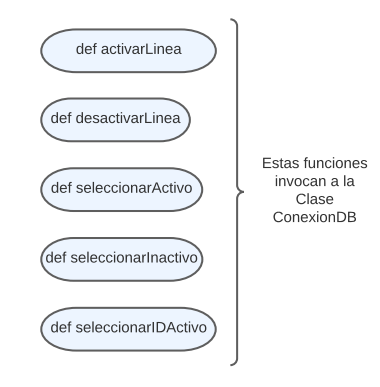
Esta clase consiste de dos métodos y cuatro funciones:

* **Métodos:**
* **def \_\_init\_\_:** Es un método constructor, cuyos parámetros son los nombres de las columnas de la tabla HistoriaMedica (idPersona, idOperacion, IdMedico, FechaHistoria, MotivoDeLaVisita, Saturacion, Operación, Precio, Tratamiento, DetalleAdicional)
* **def \_\_str\_\_:** Este método muestra los objetos del método anterior (self.idPersona, self.idOperacion, self.idMedico, etc)
* **Funciones:**
* **def guardarHistoria:** Esta función nos permitirá guardar los historiales en la clase HistoriaMedica. Los parámetros de esta función son los nombres de las columnas de la tabla HistoriaMedica. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia a los nombres de las columnas de la tabla HistoriaMedica. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox). El mensaje de la ventana emergente puede variar dependiendo si se guardó exitosamente o si hubo algún error.
* **def eliminarHistoria**: Esta función elimina permanentemente los historiales en la clase HistoriaMedica. El parámetro para esta función será “idHistorial”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia al parámetro “idHistorial” y así poder eliminarlo. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox), el mensaje de la ventana emergente puede variar dependiendo si se eliminó exitosamente o si hubo algún error.
* **def editarHistoria:** Esta función es para sobrescribir los historiales de la clase HistoriaMedica. Los parámetros para esta función serán: fechaHistoria, MotivoDeLaVisita, Operacion, Tratamiento, DetalleAdicional, Precio e idHistorial. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia cada uno de los parámetros anterior mencionados para así poder editar su contenido. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox), el mensaje de la ventana emergente puede variar dependiendo si se editó exitosamente o si hubo algún error.
* **def listarHistoria:** Esta función manda los datos a la lista listaHistoria (Tabla en la GUI). El parámetro para esta función será “idPersona”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, una variable hará referencia a la lista vacía en donde se guardarán los datos para mostrarlos (listaHistoria) y otra variable (sql) en la cual se establece que, si se cumple el parámetro where, se cumplirá el INNER JOIN. Posteriormente se ejecutará la variable “sql”, se añadirá los datos a la lista y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. Todo retorna a la listaHistoria.

**OperacionesDao.py:**

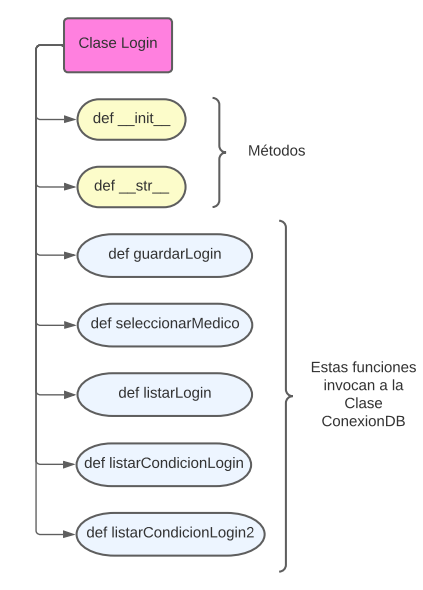
Esta clase consiste de dos métodos y cinco funciones:

* **Métodos:**
* **def \_\_init\_\_:** Es un método constructor, cuyos parámetros son los nombres de las columnas de la tabla Operaciones (Tipo, Precio)
* **def \_\_str\_\_:** Este método muestra los objetos del método anterior (self.Tipo, self.Precio)
* **Funciones:**
* **def guardarOperaciones:** Esta función nos permitirá guardar los datos en la clase Operaciones. Los parámetros de esta función son los nombres de las columnas de la tabla Operaciones. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia a los nombres de las columnas de la tabla Operaciones para así poder insertar valores. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox). El mensaje de la ventana emergente puede variar dependiendo si se guardó exitosamente o si hubo algún error.
* **def eliminarOperaciones**: Esta función elimina permanentemente los datos en la clase Operaciones. El parámetro para esta función será “idOperacion”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia al parámetro “idOperacion” y así poder eliminarlo. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (modulo messagebox), el mensaje de la ventana emergente puede variar dependiendo si se eliminó exitosamente o si hubo algún error.
* **def seleccionarIDOperacion:** Esta función permite seleccionar el tipo de operación. El parámetro para esta función será “Tipo”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, una variable hará referencia a la lista vacía en donde se guardarán los datos para mostrarlos (operacionID) y otra variable (sql) en la cual hace referencia al parámetro “Tipo”, con el fin de poder seleccionar. Posteriormente se ejecutará la variable “sql”, se añadirá los datos a la lista y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. Todo retorna a operacionID.
* **def listaOperacion:** Esta función manda los datos a una ListBOX en la GUI. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, una variable hará referencia a la lista vacía en donde se guardarán los datos para mostrarlos (listaOperacion) y otra variable (sql) en la cual se establece que, se seleccionara el tipo de operación que exista. Posteriormente se ejecutará la variable “sql”, se añadirá los datos a la lista y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. Todo retorna a la listaOperacion. En caso de presentar un error al listar, una ventana emergente aparecerá con el mensaje de error.
* **def listaPrecio:** Esta función manda los datos a una ListBOX en la GUI. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, una variable hará referencia a la lista vacía en donde se guardarán los datos para mostrarlos (listaPrecio) y otra variable (sql) en la cual se establece que, se seleccionara el precio de operación que exista. Posteriormente se ejecutará la variable “sql”, se añadirá los datos a la lista y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. Todo retorna a la listaPrecio. En caso de presentar un error al listar, una ventana emergente aparecerá con el mensaje de error.

**MedicosDao.py:**

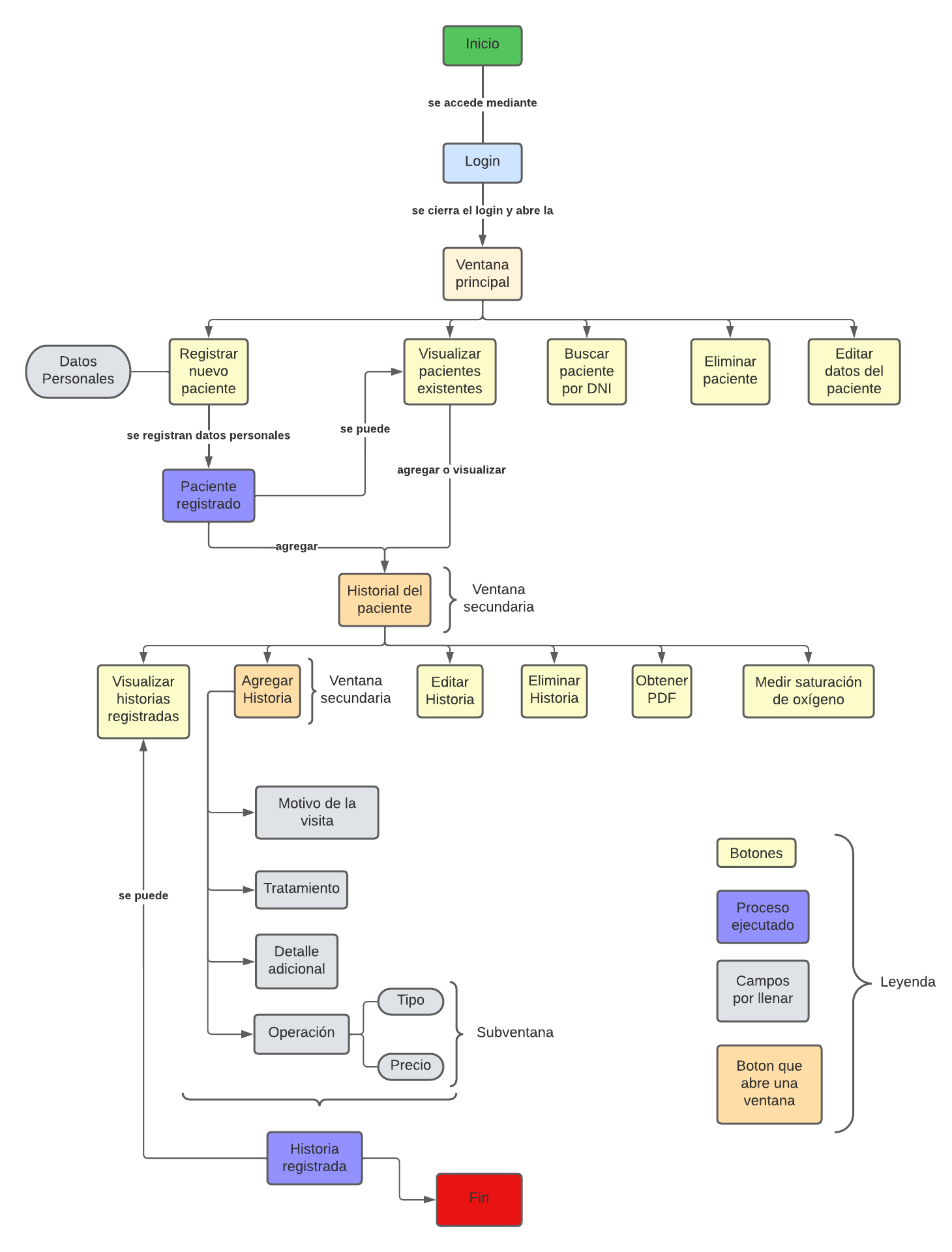
En este archivo se encuentran cinco funciones:

* **def activarLinea:** Esta función activa la línea de un médico. El parámetro de esta función es “idMedico”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia parámetro “idMedico” y así activar su línea. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. En caso de presentar un error al activar la línea, una ventana emergente aparecerá con el mensaje de error (modulo messagebox).
* **def desactivarLinea:** Esta función desactiva la línea de un médico. El parámetro de esta función es “idMedico”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia parámetro “idMedico” y así desactivar su línea. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. En caso de presentar un error al desactivar la línea, una ventana emergente aparecerá con el mensaje de error (modulo messagebox).
* **def seleccionarActivo:** Esta función selecciona el nombre del médico activo (línea = 1). Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, otra variable que hará referencia a una lista (MedicoActivo) y otra variable (sql) para hacer referencia al nombre de los médicos que se encuentran activos y así poder insertarlos en la lista. Posteriormente se ejecutará la variable “sql”, se mandarán los nombres a la lista MedicoActivo y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchone” y “cerrarConexion” respectivamente. En caso de presentar un error al listar, una ventana emergente aparecerá con el mensaje de error (modulo messagebox). Al final todo retorna a su respectiva lista.
* **def seleccionarInactivo:** Esta función selecciona el nombre del médico inactivo (línea = 0). Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, otra variable que hará referencia a una lista (MedicoInactivo) y otra variable (sql) para hacer referencia al nombre de los médicos que se encuentran inactivos y así poder insertarlos en la lista. Posteriormente se ejecutará la variable “sql”, se mandarán los nombres a la lista MedicoInactivo y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. En caso de presentar un error al listar, una ventana emergente aparecerá con el mensaje de error (modulo messagebox). Al final todo retorna a su respectiva lista.
* **def seleccionarIDActivo:** Esta función selecciona el id del médico activo (línea = 1). Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, otra variable que hará referencia a una lista (MedicoIDActivo) y otra variable (sql) para hacer referencia al id de los médicos que se encuentran activos y así poder insertarlos en la lista. Posteriormente se ejecutará la variable “sql”, se mandarán los nombres a la lista MedicoIDActivo y se cerrará la conexión con la base de datos con los métodos “execute”, “fetchall” y “cerrarConexion” respectivamente. En caso de presentar un error al listar, una ventana emergente aparecerá con el mensaje de error (modulo messagebox). Al final todo retorna a su respectiva lista.

**LoginDao.py:**

Esta clase consiste de dos métodos y cinco funciones:

* **Métodos:**
* **def \_\_init\_\_:** Es un método constructor, cuyos parámetros son los nombres de las columnas de la tabla Operaciones (Medico, Usuario, Contraseña).
* **def \_\_str\_\_:** Este método muestra los objetos del método anterior (self.Medico, self.Usuario, self.Contraseña)
* **Funciones:**
* **def guardarLogin:** Esta función nos permitirá guardar los datos de las cuentas de los médicos, además, los parámetros para esta función serán “Medico, Usuario y Contraseña”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB y otra variable (sql) para hacer referencia a los campos de la tabla Login (Medico, Usuario, Contraseña). Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Una vez ejecutado, aparecerá una ventana emergente para indicar el proceso terminado (módulo messagebox), de haberse realizado de manera correcta, de lo contrario, haciendo uso nuevamente del “módulo messagebox”, indicando que hubo un error en el proceso de registro.
* **def seleccionarMedico:** Función que selecciona un idLogin que toma el mismo valor que idMedico de la tabla Medicos para aplicarlo a la función activarLinea() o desactivarLinea() de MedicosDao.py, aquí el parámetro utilizado es “Usuario”. Se establecerá una variable (conexión) para hacer referencia a la clase ConexionDB, la lista vacía “idLogin”, la cual será llenada con la información traída desde la base de datos, gracias al método “fetchall”, y por último la variable “sql” que hace referencia al parámetro Usuario, el cual trabaja siempre y cuando el “Usuario” ingresado coincida con el de la base de datos. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Si se ha ejecutado correctamente el proceso, este finalizará, de lo contrario a través del módulo “messagebox”, aparecerá una ventana emergente para indicar que hubo un error en el proceso.
* **def listarLogin:** Función donde se establece la variable (conexión) para hacer referencia a la clase ConexionDB, la lista vacía “listaLogin”, la cual será llenada con la información traída desde la base de datos, gracias al método “fetchall”, y por último la variable “sql” que hace referencia al parámetro Login. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Si se ha ejecutado correctamente el proceso, este finalizará, de lo contrario a través del módulo “messagebox”, aparecerá una ventana emergente para indicar que hubo un error en el proceso.
* **def listarCondicionLogin:** Función que crea una lista y hace que se genere una lista que contenga las contraseñas. Aquí se establece la variable (conexión) para hacer referencia a la clase ConexionDB, la lista vacía “listarCondicionLogin”, la cual será llenada con la información traída desde la base de datos, gracias al método “fetchall”, y por último la variable “sql” que hace referencia al parámetro Contraseña. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Si se ha ejecutado correctamente el proceso, este finalizará, de lo contrario a través del módulo “messagebox”, aparecerá una ventana emergente para indicar que hubo un error en el proceso.
* **def listarCondicionLogin2:** Función que crea una lista y hace que se genere una lista que contenga los Usuarios. Aquí se establece la variable (conexión) para hacer referencia a la clase ConexionDB, la lista vacía “listarCondicionLogin2”, la cual será llenada con la información traída desde la base de datos, gracias al método “fetchall”, y por último la variable “sql” que hace referencia al parámetro Usuario. Posteriormente se ejecutará la variable “sql” y se cerrará la conexión con la base de datos con los métodos “execute” y “cerrarConexion” respectivamente. Si se ha ejecutado correctamente el proceso, este finalizará, de lo contrario a través del módulo “messagebox”, aparecerá una ventana emergente para indicar que hubo un error en el proceso.

**GUI.py:**

El diagrama presentado representa la funcionalidad de la GUI, este archivo se conecta con todas las clases y funciones anteriormente mencionadas. Se hizo uso de la librería Tkinter para darle un mejor aspecto visual, además de otros módulos, tales como: tkcalendar, datetime, fpdf, messabox, etc. Esta interfaz GUI permite interactuar con las clases y funciones antes mencionadas.

Para poder ejecutar el programa, debe hacerlo desde el archivo **PROGRAMA.py**, cabe resaltar que se requieren de ciertos módulos para poder ejecutar el programa, el Visual Studio Code señalará que módulos hacen falta.

El programa se encuentra en el siguiente repositorio Github:

<https://github.com/hdjdudixjxy/MecatronicaUNT_Prog2_DisenoDeUnSistemaDeControlParaHistorialesMedicos.git>

## Implementación de página web

En espera de más datos….

## Elaboración del Oximetro de pulso portátil

En espera de más datos….

# Capítulo 4 Resultados

En este capítulo, se discutirán los resultados obtenidos del proyecto de investigación.

## Aplicación de escritorio

## Página web

## Oximetro de pulso portátil

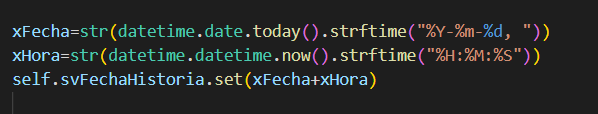
# Capítulo 5 Conclusiones y Recomendaciones

## Conclusiones

## Recomendaciones

* Al momento de utilizar datetime y querer la hora, junto con la fecha, usar strftime, pero no de forma directa: datetime.date.today().strftime("%Y-%m-%d, %H:%M:%S ")

Sino separar la fecha con today y la hora con now, debido a que en nuestro proyecto generó un error inesperado. Luego solo agregarlos a variables y concatenarlas para poder mostrarlas.



* Cuando se ingresa el DNI, si el número empieza en 0, se elimina y en la base de datos, tanto como en el treeview de la interfaz se ven 6 números, no 7, debido a el uso de tk.StringVar en casi todas las variables de los entrys. En este caso es recomendable usar tk.IntVar para números.
* El volver a cada momento los ID a None es muy importante ya que son las instancias que más se usan en todo el proyecto, tanto para DatosPaciente e HistoriaMedica, esto se debe a que toman muchos valores en cada función y para que Python no genere error es que se debe agregar este valor vacío al ID para que pueda tomar uno nuevo consecutivamente cada vez que se haga uso de una función al presionar los botones en la interfaz.
* El método pack si se sabe manejar correctamente, puede ser mejor opción para ubicar los widgets de Tkinter, pero por simplicidad se utilizó grid, lo cual redució notablemente las líneas de código. Con el método pack no se distorsiona tanto como con grid al maximizar la ventana y es por eso que utilizamos.



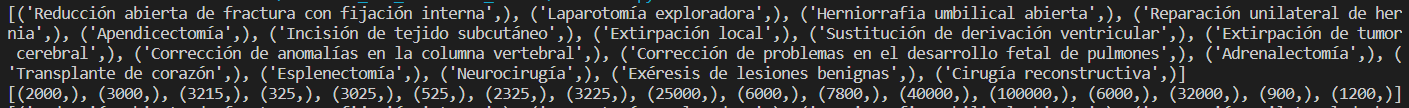
* Al momento de concatenar en MySQL, utilizamos CONCAT().



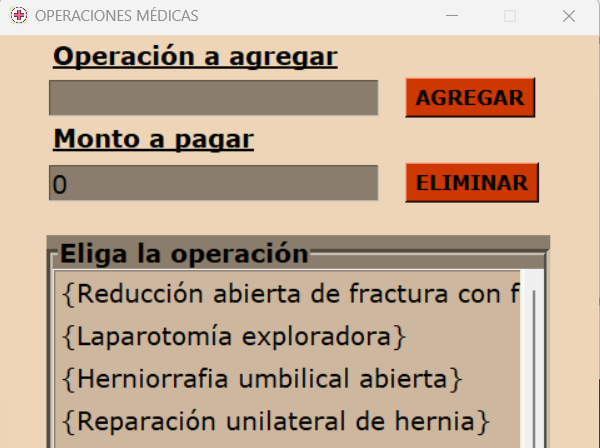
* Como a def listarOperacion(): le colocamos un

SELECT Operacion2 FROM Tipo WHERE TRUE

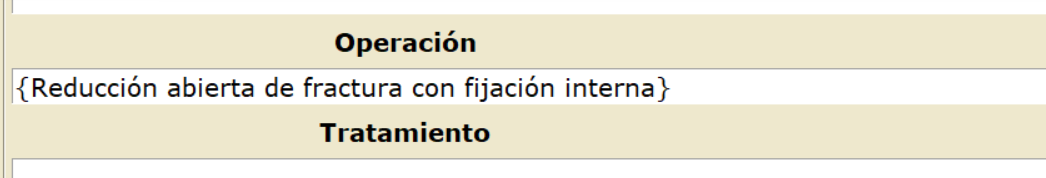
Al utilizar List(cursor.fetchall()) lo que nos devuelve es una lista cuyos elementos son tuplas y no strings:



Ya que no tomamos elementos en una sola fila, sino una sola columna y esto es un grave problema, porque al aplicar insert en nuestro ListBox sale de la siguiente manera:



Y así aparecería en el TopLevel, dando un error al insertarlo así, se tendría que borrar de forma manual las llaves y no es una buena alternativa para el usuario.



La solución fue ejecutar nuestra función listar normalmente y luego esos datos almacenarlos en una nueva lista (ListaOperacionTRUE), donde se transformen de tupla a string

self.ListaOperacion=listarOperacion()

self.ListaOperacionTRUE=[]

Creamos de igual manera una ListaOperacion2 que contendrá de forma individual por cada iteración el elemeno en la posición p de la ListaOperacion y una vez hecho esto, como posee un solo elemento, en su posición 0 la agregamos a la ListaOperacionTRUE

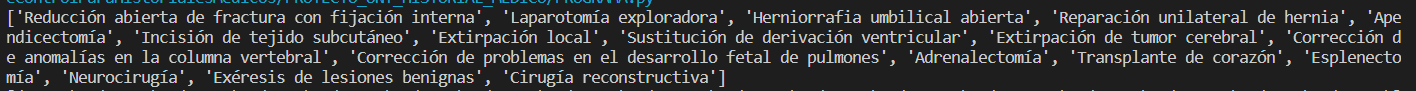
for p in range(len(self.ListaOperacion)):

      self.ListaOperacion2=self.ListaOperacion[p]

      self.lista.insert(tk.END,self.ListaOperacion2[0])

      self.ListaOperacionTRUE.append(self.ListaOperacion2[0])

De esta forma se cambiaría todos los datos de tuplas a strings:



De igual manera con la ListaPrecio a ListaPrecioTRUE:





# Referencias Bibliográficas

EsSalud. (26 de Septiembre de 2019). *Implementa historia clínica digital para atención de asegurados*. Obtenido de EsSalud: http://noticias.essalud.gob.pe/?publicacion=implementa-historia-clinica-digital-para-atencion-de-asegurados

Fernandez, R. (4 de Enero de 2021). *¿Qué es Tkinter y por qué utilizarlo?* Obtenido de Unipython: https://unipython.com/tkinter-introduccion/

Flores, F. (13 de Mayo de 2022). *¿Qué es Visual Studio Code y qué ventajas ofrece?* Obtenido de OpenWebinars: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/

RPP Noticias. (2 de Febrero de 2022). *Historia clínica eléctronica: Cinco ventajas de tener un sistema digital en todos los centros de salud*. Obtenido de RPP Noticias: https://rpp.pe/campanas/valor-compartido/historia-clinica-electronica-cinco-ventajas-de-tener-un-sistema-digital-en-todos-los-centros-de-salud-el-pais-que-queremos-noticia-1383397?ref=rpp

Santander Universidades. (1 de Marzo de 2022). *¿Qué es Python?* Obtenido de Becas Santander: https://www.becas-santander.com/es/blog/python-que-es.html

Sanunga Totoy, J. E., & Pérez Palma, K. N. (28 de Enero de 2019). *Repositorio Institucional de la Universidad Politécnica Salesiana: Implementación del sistema para el control de historia clínica de pacientes en centro odontológico Dental Group*. Obtenido de DSpace: https://dspace.ups.edu.ec/handle/123456789/16767

Silvia Velito, A., & Tejada Soriano, S. R. (2010). *La historia clinica como instrumento de calidad*. Obtenido de autoriamedicahoy.com: http://www.auditoriamedicahoy.com/biblioteca/La%20historia%20cl%C3%ADnica%20como%20instrumento%20de%20calidad%20Tejada%20Velito.pdf

Yrinna Benites, K. A. (19 de Noviembre de 2016). ANÁLISIS Y DISEÑO DE PROTOTIPO DE SOFTWARE PARA LA AUTOMATIZACIÓN DE HISTORIAS CLÍNICAS DEL POLICLÍNICO UDEP. Piura, Perú.

**REFERENCIAS LINKOGRÁFICAS**

ANACONDA. (2018). Understanding Conda and Pip. Retrieved from https://www.anaconda.com/blog/understanding-conda-and-pip

Python. (2022). tkinter — Interface de Python para Tcl/Tk. Retrieved from https://docs.python.org/es/3/library/tkinter.html

Estrada web group. (2022). ¿Qué es el tipo de dato varchar de SQL Server y cuando utilizarlo? Retrieved from https://estradawebgroup.com/Post/-Que-es-el-tipo-de-dato-varchar-de-SQL-Server-y-cuando-utilizarlo-/20355

SALVADOR, U. D. EL. (2019). Introducción a SQLITE. Retrieved from https://eisi.fia.ues.edu.sv/materialpublico/pdm115/2019/labs/PDM115\_guia\_lab03\_SQLite.pdf

Runebook.dev. (n.d.). Tipos de datos en SQLITE. Retrieved from https://runebook.dev/es/docs/sqlite/datatype3

TUTORIAL, P. (2022). Tkinter Object-Oriented Window. Retrieved from https://www.pythontutorial.net/tkinter/tkinter-object-oriented-window/

Guia Tkinter. (2015). Interfaz gráfica con Tkinter. Retrieved from https://guia-tkinter.readthedocs.io/es/develop/index.html

FJSevilla. (2020). Fuentes disponibles en Tkinter. Retrieved from https://es.stackoverflow.com/questions/330481/fuentes-disponibles-en-tkinter

Python. (2022). Envoltorio de fuente Tkinter. Retrieved from https://docs.python.org/es/3.9/library/tkinter.font.html

Colors-symbolic color names recognized by Tk. (2018). Retrieved from https://www.tcl.tk/man/tcl8.5/TkCmd/colors.html

Patricia. (2019). Tabla de colores Tkinter. Retrieved from http://patriciaemiguel.com/python/2019/08/01/python-tkinter-colores.html

SQLite. (2022). SQLite. Retrieved from https://www.sqlite.org/index.html

Anaconda / packages / sqlite. (2022). Retrieved from https://anaconda.org/anaconda/sqlite

Python 3 para impacientes. (2016). Retrieved from https://python-para-impacientes.blogspot.com/2016/02/variables-de-control-en-tkinter.html

Mizipzor. (2009). Understanding Python super() with \_\_init\_\_() methods [duplicate]. Retrieved from https://stackoverflow.com/questions/576169/understanding-python-super-with-init-methods

Educative.io. (n.d.). What is super() in Python? Retrieved from https://www.educative.io/answers/what-is-super-in-python

Dev Prakash, S. (2021). How to Disable / Enable a Button in TKinter? Retrieved from https://www.tutorialspoint.com/how-to-disable-enable-a-button-in-tkinter#:~:text=Tkinter Button widgets can be,and disabling the button%2C respectively.

tutorialesprogramacion. (n.d.). Tipos de datos básicos para definir los campos de una tabla. Retrieved from https://www.tutorialesprogramacionya.com/sqliteya/detalleconcepto.php?punto=4&codigo=4&inicio=0

JC Chouinard. (2020). if \_\_name\_\_ == ‘\_\_main\_\_’: What does it mean (Python). Retrieved from https://www.jcchouinard.com/python-if-name-equals-main/#:~:text=When you start working with,What does it mean%3F&text=What this function do is,the module from another one.%0A

Python. (2022). tkinter.messagebox. Retrieved from https://docs.python.org/3/library/tkinter.messagebox.html

Python. (n.d.). tkinter.ttk — Tk widgets temáticos. Retrieved from https://docs.python.org/es/3/library/tkinter.ttk.html%0A

Python. (n.d.). tkinter.ttk — Tk widgets temáticos. Retrieved from https://stackoverflow.com/questions/19561727/what-is-the-difference-between-the-widgets-of-tkinter-and-tkinter-ttk-in-python

recursos python. (2017). Posicionar elementos en Tkinter. Retrieved from https://recursospython.com/guias-y-manuales/posicionar-elementos-en-tkinter/

Manuel, G. (2022). Python-Tkinter. Retrieved from https://www.youtube.com/watch?v=Yvs7YFmKwuE&list=PLh7JzoyIyU4LKz9h3KC7VNrKPdkm0o8N4&index=20&ab\_channel=ManuelGonzález%0A

RIP Tutorial. (n.d.). Difference between Tk and Toplevel. Retrieved from https://riptutorial.com/tkinter/example/22130/difference-between-tk-and-toplevel

Python. (2022). *Funciones generales relacionadas con el calendario*. 28 de Agosto. https://docs.python.org/es/3/library/calendar.html

pypi. (2019). *Tkcalendar 1.6.1*. 28 de Diciembre. https://pypi.org/project/tkcalendar/

J2LOGO. (n.d.). *\*args y \*\*kwargs en Python. Una explicación y ejemplos de uso.* https://j2logo.com/args-y-kwargs-en-python/

Python. (n.d.). *Tipos básicos de fecha y hora*. https://docs.python.org/es/3/library/datetime.html#module-datetime

Programación-y-más. (2021). *¿Cómo funciona INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN?* https://programacionymas.com/blog/como-funciona-inner-left-right-full-join

Anaconda.org. (2021). *conda-forge / packages / fpdf*. Enero. https://anaconda.org/conda-forge/fpdf

*recursos python*. (2017). https://recursospython.com/guias-y-manuales/lista-listbox-en-tkinter/

*Satck Overflow*. (2021). https://stackoverflow.com/questions/47954758/how-to-get-a-horizontal-scrollbar-in-tkinter

HEKTOR DOCS. (2018). *Dialogs*. https://docs.hektorprofe.net/python/interfaces-graficas-con-tkinter/dialogs-dialogos/

python. (2022). *webbrowser*. https://docs.python.org/3/library/webbrowser.html

PythonCode. (2022). *How to Control your Mouse in Python*. https://www.thepythoncode.com/article/control-mouse-python

# ANEXOS