**Python 3 - Nivel Inicial**

### Ing. Juan Barreto

Entrega Final

**Alcance de la Aplicación:**

Es una aplicación de gestión de citas médicas que utiliza una base de datos SQLite para almacenar los datos de los pacientes y sus citas médicas. Este código implementa una interfaz gráfica utilizando la biblioteca Tkinter.

El código crea una ventana principal donde se muestran diferentes campos de entrada y botones para agregar, editar y borrar citas médicas. Además, posee un Treeview en el que se muestran los datos de las citas médicas almacenadas en la base de datos.

La aplicación utiliza funciones de validación de entrada para asegurarse de que se introduzcan datos válidos en los campos de entrada. Cuando se agregan o editan datos, se actualiza el Treeview para mostrar los datos actualizados.

En resumen, esta aplicación es una herramienta de gestión de citas médicas que utiliza una base de datos para almacenar y mostrar datos.

Documentación

**Backend:**

Este código importa diferentes módulos y bibliotecas de Python que se utilizarán en el programa:

from tkinter import \*

-Este comando importa el módulo principal de Tkinter, que es una biblioteca gráfica para crear interfaces de usuario en Python.

from tkinter import ttk

-Este comando importa el submódulo ttk de Tkinter, que proporciona widgets adicionales y más avanzados que el módulo principal.

from tkinter.messagebox import \*

-Este comando importa la función messagebox de Tkinter, que se utiliza para mostrar ventanas emergentes con mensajes de aviso, confirmación, información, etc.

import tkinter as tk

-Este comando importa el módulo principal de Tkinter y lo renombra como 'tk', lo que permite usarlo de manera más concisa en el resto del programa.

import sqlite3

-Este comando importa el módulo sqlite3 de Python, que proporciona una interfaz para trabajar con bases de datos SQLite en Python.

import os

-Este comando importa el módulo os de Python, que proporciona una interfaz para interactuar con el sistema operativo subyacente y realizar operaciones como crear, eliminar y mover archivos y carpetas.

from PIL import ImageTk, Image

-Este comando importa el módulo ImageTk y la clase Image de la biblioteca PIL (Python Imaging Library), que proporciona herramientas para abrir, manipular y guardar imágenes en Python.

import re

-Este comando importa el módulo re de Python, que proporciona funciones para trabajar con expresiones regulares.

BASE\_DIR = os.path.dirname((os.path.abspath(\_\_file\_\_)))

-Esta línea de código define una variable llamada BASE\_DIR que se utiliza para almacenar la ruta absoluta del directorio actual en el que se encuentra el archivo de Python en ejecución. Para ello, utiliza la biblioteca os para obtener la ruta del directorio del archivo actual.

ruta = os.path.join(BASE\_DIR, "logo1.png")

-Esta línea de código define una variable llamada ruta que se utiliza para almacenar la ruta absoluta de un archivo llamado "logo1.png". Para ello, utiliza la biblioteca os para concatenar la ruta del directorio actual almacenada en la variable BASE\_DIR con el nombre del archivo.

master = Tk()

-Esta línea de código crea una nueva ventana de Tkinter y la almacena en una variable llamada 'master'. La ventana se utiliza como contenedor principal para todos los widgets de la aplicación.

master.title("Appointment System")

-Esta línea de código establece el título de la ventana principal como "Appointment System".

master["background"] = "#F7F3F3"

-Esta línea de código establece el fondo de la ventana principal como un color de fondo en formato hexadecimal (#F7F3F3). Esto cambia el color de fondo de la ventana principal para que sea más legible y atractivo visualmente.

def conexion():

-Esta línea de código define una nueva función llamada conexion().

con = sqlite3.connect("baseDeDatos.db")

-Esta línea de código utiliza el módulo sqlite3 para conectarse a una base de datos SQLite que se encuentra en el mismo directorio que el archivo de Python y se llama "baseDeDatos2.db". La función connect() devuelve un objeto de conexión que se almacena en la variable con.

return con

-Esta línea de código devuelve el objeto de conexión con al llamar a la función conexion(). Esto permite que el objeto de conexión sea utilizado en otras partes del código para realizar operaciones en la base de datos SQLite, como ejecutar consultas SQL o insertar nuevos datos.

def crear\_tabla(con):

-Esta línea de código define una nueva función llamada crear\_tabla() que recibe un objeto de conexión con como parámetro.

cursor = con.cursor()

-Esta línea de código crea un objeto de cursor que se utiliza para ejecutar consultas SQL en la base de datos. El objeto de cursor se crea a partir del objeto de conexión con que se pasó como parámetro a la función.

sql = "CREATE TABLE IF NOT EXISTS pacientes (id integer PRIMARY KEY AUTOINCREMENT, paciente text, especialidad text, dia text, hora text, obraSocial text)"

-Esta línea de código define una consulta SQL para crear una nueva tabla llamada "pacientes" en la base de datos. La tabla tiene seis columnas: "id", "paciente", "especialidad", "dia", "hora" y "obraSocial". La columna "id" se establece como clave primaria y se utiliza el modificador "AUTOINCREMENT" para que se genere automáticamente un nuevo valor para cada nuevo registro.

cursor.execute(sql)

-Esta línea de código ejecuta la consulta SQL definida en la variable sql utilizando el objeto de cursor creado anteriormente.

   con.commit()

-Esta línea de código confirma los cambios realizados en la base de datos mediante la función commit(). Esto asegura que los cambios se guarden permanentemente en la base de datos.

con = conexion()

crear\_tabla(con)

-Estas líneas de código crean una nueva conexión a la base de datos utilizando la función conexion() y almacenan el objeto de conexión en la variable con. Luego, llaman a la función crear\_tabla() pasando el objeto de conexión con como parámetro para crear la tabla "pacientes" en la base de datos.

def mostrar(con):

Acá se define una función llamada mostrar() que se utiliza para recuperar todos los registros de una tabla llamada "pacientes" en una base de datos SQLite y mostrar la información en la consola de Python. También se define un bloque try y except que llama a las funciones conexion() y crear\_tabla() para asegurarse de que la tabla "pacientes" exista antes de intentar mostrar su contenido.

  cursor = con.cursor()

-Esta línea de código define una nueva función llamada mostrar() que recibe un objeto de conexión con como parámetro.

sql = "SELECT \* FROM pacientes"

-Esta línea de código crea un objeto de cursor que se utiliza para ejecutar consultas SQL en la base de datos. El objeto de cursor se crea a partir del objeto de conexión con que se pasó como parámetro a la función.

  cursor.execute(sql)

-Esta línea de código define una consulta SQL para seleccionar todos los registros de la tabla "pacientes" en la base de datos.

resultado = cursor.fetchall()

-Esta línea de código ejecuta la consulta SQL definida en la variable sql utilizando el objeto de cursor creado anteriormente.

for i in resultado:

        print(i[2])

-Esta línea de código recupera todos los registros seleccionados por la consulta SQL y los almacena en la variable resultado como una lista de tuplas.

        tree.insert(

            "",

            "end",

            text=i[0],

            values=(

                i[1],

                i[2],

                i[3],

                i[4],

                i[5]

            ),

        )

Este bloque de código itera sobre todos los registros recuperados de la tabla "pacientes" y los muestra en la consola de Python. También inserta cada registro como una nueva fila en un widget de Treeview llamado tree, que se asume que ya está definido en otro lugar del código. Cada columna de la fila se establece utilizando los valores de las tuplas almacenados en la variable resultado.

try:

    conexion()

    crear\_tabla()

except:

    #se imprime en la consola

    print("Ya existe la tabla, puede editar")

-Este bloque de código intenta crear una nueva conexión a la base de datos y crear la tabla "pacientes" utilizando las funciones conexion() y crear\_tabla(), respectivamente. Si la tabla ya existe, la excepción except captura el error y muestra un mensaje en la consola de Python. Esto se hace para asegurarse de que la tabla "pacientes" exista antes de intentar mostrar su contenido en la consola o en cualquier otro widget del programa.

def funcion\_a(paciente, especialidad, dia, hora, obraSocial, tree):

Acá se define una función llamada funcion\_a() que se utiliza para validar y agregar nuevos registros a una tabla llamada "pacientes" en una base de datos SQLite. La función toma varios parámetros, incluyendo información del paciente, la especialidad, el día y la hora de la cita, la obra social, y un objeto de Treeview.

 cadena = paciente

 patron= "^[A-Za-z]+(?i:[\_-]+)\*$"

-Esta línea de código define una nueva función llamada funcion\_a() que recibe los parámetros paciente, especialidad, dia, hora, obraSocial, y tree.

 print(paciente, especialidad, dia, hora, obraSocial)

Estas líneas de código definen una cadena de texto cadena que se utilizará para validar el nombre del paciente. La variable patron define una expresión regular que se utiliza para validar el nombre del paciente. En este caso, la expresión regular valida que el nombre solo contenga letras y que no contenga espacios al principio o al final.

    if(re.match(patron, cadena)):

-Esta línea de código utiliza la función match() del módulo re para verificar si el nombre del paciente cumple con el patrón definido en la variable patron.

  con=conexion()

        cursor = con.cursor()

        data = (paciente, especialidad, dia, hora, obraSocial)

        sql =  "INSERT INTO pacientes (paciente, especialidad, dia, hora, obraSocial) VALUES (?,?,?,?,?)"

        cursor.execute(sql, data)

        con.commit()

-Estas líneas de código crean una conexión a la base de datos y crean un objeto de cursor que se utiliza para agregar un nuevo registro a la tabla "pacientes". Los datos del nuevo registro se pasan como una tupla llamada data. La consulta SQL utilizada para agregar el nuevo registro se define en la variable sql. Luego, la función execute() del objeto de cursor se utiliza para ejecutar la consulta SQL con los datos proporcionados. Finalmente, la conexión se confirma con la función commit() para guardar los cambios en la base de datos.

showinfo("Agregar", "Turno añadido")

actualizar\_treeview(tree)

-Estas líneas de código muestran un mensaje de información utilizando la función showinfo() del módulo messagebox y actualizan el objeto Treeview pasado como parámetro utilizando la función actualizar\_treeview(), que se asume que está definida en otro lugar del código.

else:

        print("error en carga")

-Si el nombre del paciente no cumple con el patrón definido en la variable patron, se imprime un mensaje de error en la consola de Python.

def funcion\_b(paciente, especialidad, dia, hora, obraSocial, tree):

Acá se define una función llamada funcion\_b() que se utiliza para editar los registros a una tabla llamada "pacientes" en una base de datos SQLite. La función toma varios parámetros, incluyendo información del paciente, la especialidad, el día y la hora de la cita, la obra social, y un objeto de Treeview.

cadena = paciente

patron="^[A-Za-záéíóú]\*$"

-Esta línea de código define una nueva función llamada funcion\_a() que recibe los parámetros paciente, especialidad, dia, hora, obraSocial, y tree.

if(re.match(patron, cadena)):

-Esta línea de código utiliza la función match() del módulo re para verificar si el nombre del paciente cumple con el patrón definido en la variable patron.

 valor = tree.selection()

 print(valor)

item = tree.item(valor)

print(item)

Alojamos en la variable Valor e Item, la línea de la tabla seleccionada. Para posteriormente imprimir los valores correspondientes por consola.

  cursor = con.cursor()

-Esta línea de código define una nueva función llamada cursor que recibe un objeto de conexión con como parámetro.

 data = (

        entryPaciente.get(),

        entryEspecialidad.get(),

        entryDia.get(),

        entryHora.get(),

        entryObraSocial.get(),

        str(item["text"]),

    )

En Data utilizamos la función .get() para ingresar los nuevos datos que necesitamos editar, en cada entry respectivamente. Y finalizamos diciéndole que en cada ítem tendrmos un texto, es decir una variable str.

sql = "UPDATE pacientes SET paciente = ?, especialidad = ?, dia = ?, hora = ?, obraSocial = ? WHERE id = ?"

print(item)

Le decimos la base de datos SQL que realice un UPDATE a la tabla pacientes, donde las columnas que ha que editar son: paciente, especialidad, día, hora, obraSocial. Para luego imprimir por consola los valores que posee ítem.

cursor.execute(sql)

-Esta línea de código ejecuta la consulta SQL definida en la variable sql utilizando el objeto de cursor creado anteriormente.

con.commit()

-Esta línea de código confirma los cambios realizados en la base de datos mediante la función commit(). Esto asegura que los cambios se guarden permanentemente en la base de datos.

showinfo("Editar", "Turno Modificado")

actualizar\_treeview(tree)

-Estas líneas de código muestran un mensaje de información utilizando la función showinfo() del módulo messagebox y actualizan el objeto Treeview pasado como parámetro utilizando la función actualizar\_treeview(), que se asume que está definida en otro lugar del código.

else:

        print("error en carga")

-Si el nombre del paciente no cumple con el patrón definido en la variable patron, se imprime un mensaje de error en la consola de Python.

def funcion\_c(tree):

Acá se define una función llamada funcion\_c() que se utiliza para borrar registros de la tabla llamada "pacientes" en una base de datos SQLite.

 valor = tree.selection()

 print(valor)

 item = tree.item(valor)

 print(item)

print(item['text'])

Alojamos en la variable valor e item los datos seleccionados con Tree. Y los imprimimos por terminal para visualizarlos.

mi\_id = item['text']

El texto que estaba guardado en la variable ítem anterior lo alojamos en mi\_id

showinfo("Borrar", "Registro borrado")

Mostramos en una ventana emergente que el registro se a eliminado correctamente.

 cursor=con.cursor()

data = (mi\_id,)

    sql = "DELETE FROM pacientes WHERE id = ?;"

    cursor.execute(sql, data)

    con.commit()

-Estas líneas de código crean una conexión a la base de datos y elimina el objeto seleccionado por el cursor, borrando registro a la tabla "pacientes". Los datos eliminados se pasan como una tupla llamada data. Luego, la función execute() del objeto de cursor se utiliza para ejecutar la consulta SQL con los datos proporcionados. Finalmente, la conexión se confirma con la función commit() para guardar los cambios en la base de datos.

tree.delete(valor)

Finalizamos la función diciendo el valor que se debe eliminar de la tabla.

def actualizar\_treeview(mitreview):

    records = mitreview.get\_children()

Esta nueva función se utiliza para actualizar todos los datos que se agregaron, editaron o eliminaron de la base de datos. Para después obtenerlos y guardalos en mi treview.

sql = "SELECT \* FROM pacientes ORDER BY id ASC"

    con=conexion()

    cursor=con.cursor()

    datos=cursor.execute(sql)

En la primera sentencia selecionamos todos los dato de la tabla Pacientes ordenados por orden ascendente. Luego ejecuta la consulta SQL definida en la variable sql utilizando el objeto de cursor creado anteriormente.

resultado = datos.fetchall()

    for fila in resultado:

        print(fila)

        mitreview.insert("", 0, text=fila[0], values=(fila[1], fila[2], fila[3],fila[4],fila[5]))

Los resultados de datos de las filas los imprimimos por terminal y luego insertamos en mi treview cada dato actualizado en su respectiva fila.

titulo = Label(master, text="INGRESE SUS DATOS", bg="#D83737", fg="thistle1", height=1, width=60)

titulo.grid(row=0, column=0, columnspan=6, padx=2, pady=2, sticky=W+E)

Definimos una variable titulo que nos muestre en nuestro master el texto definido. A su vez determinamos una serie de formatos deseados como el color del fondo, el color de mismo, el color de la letra. Como asi también el tamaño que necesitamos y la posición.

entryPaciente, entryEspecialidad, entryDia, entryHora, entryObraSocial = StringVar(),StringVar(), IntVar(), StringVar(),StringVar()

w\_ancho = 20

Definimos las variables de entrada como entry y establecemos que tipo de dato va a ingresar por cada input. Para finalizar le decimos el ancho del campo.

paciente = Label(master, text="Paciente:")

paciente.grid(row=1, column=0, sticky=W,)

entryPaciente = Entry(master, textvariable=entryPaciente, width=70)

entryPaciente.grid(row=1, column=1)

especialidad = Label(master, text="Especialidad:")

especialidad.grid(row=2, column=0, sticky=W)

entryEspecialidad = Entry(master, textvariable=entryEspecialidad, width=70)

entryEspecialidad.grid(row=2, column=1)

dia = Label(master, text="Día:")

dia.grid(row=3, column=0, sticky=W)

entryDia = Entry(master, textvariable=entryDia, width=50)

entryDia.grid(row=3, column=1)

hora = Label(master, text="Hora:")

hora.grid(row=4, column=0, sticky=W)

entryHora = Entry(master, textvariable=entryHora, width=50)

entryHora.grid(row=4, column=1)

obraSocial = Label(master, text="Obra Social:")

obraSocial.grid(row=5, column=0, sticky=W)

entryObraSocial = Entry(master, textvariable=entryObraSocial, width=70)

entryObraSocial.grid(row=5, column=1)

, width=70)

entryPaciente.grid(row=1, column=1)

Definimos que contiene en la variables pacientes, especialidad, día, hora y obraSocial. Los datos que contienen los mismos ingresaran en un Label en la ventana master. También mostraran un texto con el nombre de cada variable para que el usuario sepa que dato debe ingresar. También se establece un formato específico y una ubicación para este input. Seguimos con la variable entry que nos permite ingresar información por la interfaz, también con un formato determina. Y finalizamos con la ubicación de este input de ingreso de datos.

image2 = Image.open(ruta)

image1 = ImageTk.PhotoImage(image2)

background\_label = tk.Label(master, image=image1)

background\_label.place(x=600, y=30)

En la primera línea de sitaxis creamos una variable Imagen2 y la enlazamos con un ruta determianda y la abrimos. En la variable Image1 utilizamos la librería imageTK.PhotoImage para alijar Image2. Finalizamos diciendo cual es color de fondo, tamaño y posición de la misma.

tree = ttk.Treeview(master)

Definimos la variable Tree y utilizamos la librería ttk de treeview en la ventana master.

tree["columns"] = ("col1", "col2", "col3", "col4", "col5")

En la variable tree creamos 5 columnas que son las que utilizaremos para alojar la información que usa la aplicación.

tree.column("#0", width=90, minwidth=50, anchor=W)

tree.column("col1", width=50, minwidth=50)

tree.column("col2", width=50, minwidth=50)

tree.column("col3", width=200, minwidth=80)

tree.column("col4", width=200, minwidth=80)

tree.column("col5", width=200, minwidth=80)

Definimos las dimensiones y la ubicación de las 5 columnas establecidas con anterioridad. Destaco que la columna que contiene Id es de menor tamaño que las otras.

tree.heading("#0", text="ID")

tree.heading("col1", text="Paciente    ")

tree.heading("col2", text="Especialidad")

tree.heading("col3", text="Fecha       ")

tree.heading("col4", text="Hora        ")

tree.heading("col5", text="O.Social    ")

Las columnas definidas tiene un head o encabezado con los nombres que se detallan: ID, Paciente, Especialidad, Fecha, Hora y O. Social.

tree.grid(row=12, column=0, columnspan=4)

Todas estas columnas se posicionan en la ubicación 0 y poseen ciertos formato especificado aquí.

s = ttk.Style()

s.configure("Peligro.TButton", foreground="#ff0000")

s.map("Peligro.TButton", foreground=[("active", "#7DCEA0")])

en la variable S llamamos a ttk.Style, la cual es una función para configurar determinados CSS a los botones que usaremos. Los mismos se llaman Peligro.TButton, poseen determinado color de fondo y al ser activados cambian a otro color especificado.

botonagregar = ttk.Button(master, text="Agregar", style="Peligro.TButton", command=lambda: funcion\_a(entryPaciente.get(), entryEspecialidad.get(), entryDia.get(), entryHora.get(), entryObraSocial.get(), tree))

botonagregar.grid(row=20, column=1, sticky=E+W)

botonEditar = ttk.Button(master, text="Editar", style="Peligro.TButton", command=lambda: funcion\_b(entryPaciente.get(), entryEspecialidad.get(), entryDia.get(), entryHora.get(), entryObraSocial.get(), tree))

botonEditar.grid(row=20, column=0, sticky=E)

botonborrar = ttk.Button(master, text="Borrar", style="Peligro.TButton", command=lambda: funcion\_c(tree))

botonborrar.grid(row=20, column=2, sticky=W)

Aquí declaramos los tres botones que usaremos en la ventana master, los cuales se llaman Agregar, Editar y Borrar. Los tres utilizan una función lambda y llaman a tres funciones diferentes, Agregar a la función\_a, Editar a la función\_b y Borrar a la función\_c. posteriormente se definen los style de cada uno y los entry que se agregaran, editaran o borraran. Finalizamos con el formato que utilizara cada botón y su correspondiente ubicación.

titulo2 = Label(master, bg="#D83737", fg="thistle1", height=1, width=60)

titulo2.grid(row=30, column=0, columnspan=6, padx=2, pady=2, sticky=W+E)

titulo3 = Label(master, bg="#D83737", fg="thistle1", height=1, width=60)

titulo3.grid(row=18, column=0, columnspan=6, padx=2, pady=2, sticky=W+E)

Creamos unas líneas decorativas con determinado color de fondo, ancho, alto y ubicación. A los fines de realizar una separación visual de los elementos en la ventana master.

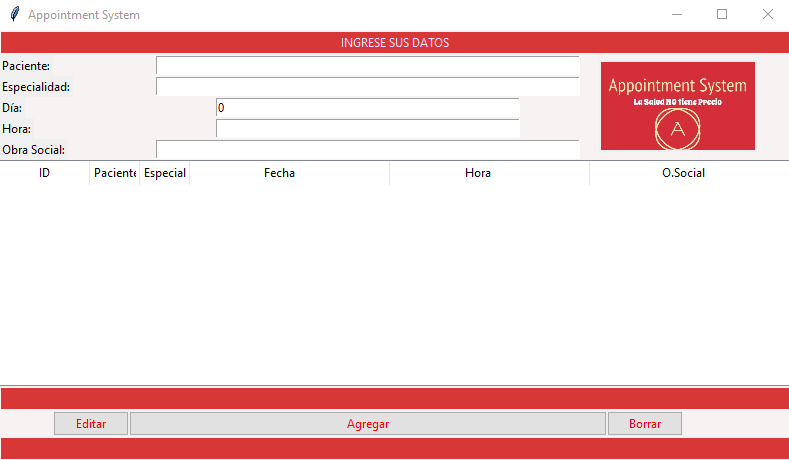
mostrar(con)

master.mainloop()

Finalizamos el Backend de la aplicación con un bucle de ejecución y una impresión de los datos.

Documentación

**Frontend:**

Para comenzar realizaremos una presentación integral de la interfaz “Appointmnet System”. Y continuaremos detallando las partes de la misma y sus funcionalidades

Encabezado:

Subtitulo informativa:

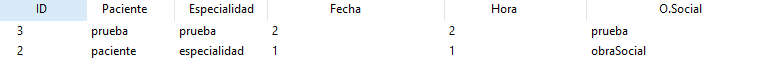
Datos necesarios a completar:

* Primero tenemos los nombres que nos especifican el dato que se debe ingresar
* Segundo tenemos el campo donde se debe escribir la información necesaria para el turno



ISO logotipo de la aplicación:



Tabla con información cargada en la Base de Datos:

Botonera con las acciones que se pueden realizar en el aplicativo:



Ventanas emergentes que confirman las acciones realizadas:

