Python TKinter

Marco Teórico: 1.1 ¿Qué entiende por GUI? GUI (Graphical User Interface) se refiere a una interfaz gráfica de usuario que permite la interacción entre el usuario y un programa mediante elementos visuales como ventanas, botones, menús, campos de entrada, entre otros.

1.2 Describir 3 Framework de GUI para Python. Además, que se usen en C++.

Tkinter: Es el framework de GUI estándar incluido en la biblioteca estándar de Python. Proporciona un conjunto de widgets y herramientas para construir interfaces gráficas de usuario.

PyQt: Es un conjunto de enlaces de Python para la biblioteca Qt de C++. Permite crear interfaces gráficas de usuario altamente personalizables y profesionales.

wxPython: Es un enlace de Python para la biblioteca wxWidgets escrita en C++. Proporciona una amplia variedad de widgets y herramientas para desarrollar aplicaciones con interfaces gráficas de usuario.

1.3 ¿Qué diferencia tienen los módulos TK y TTK? Los módulos TK y TTK en Tkinter se refieren a conjuntos de widgets disponibles para construir interfaces gráficas de usuario.

TK: Es el conjunto de widgets nativos de Tkinter. Estos widgets tienen un aspecto nativo del sistema operativo en el que se ejecuta la aplicación.

TTK (Themed Tk): Es un conjunto de widgets temáticos introducidos en Tkinter a partir de la versión 8.5 de Tcl/Tk. Estos widgets tienen un aspecto más moderno y uniforme en diferentes sistemas operativos. TTK también proporciona algunos widgets adicionales que no están presentes en el conjunto TK.

1.4 ¿Qué es un Widget? Dar 5 ejemplos. Un widget es un elemento interactivo o visual en una interfaz gráfica de usuario. Representa una funcionalidad específica y puede incluir botones, etiquetas, campos de entrada, listas desplegables, etc.

Ejemplos de widgets en Tkinter:

Button (Botón): Permite al usuario hacer clic para realizar una acción.

Label (Etiqueta): Muestra texto o una imagen sin capacidad de interacción.

Entry (Campo de entrada): Permite al usuario ingresar texto.

ComboBox (Lista desplegable): Proporciona una lista de opciones para que el usuario elija una.

Checkbutton (Casilla de verificación): Permite al usuario seleccionar una o varias opciones.

1.5 ¿Explicar el Widget Notebook en Tkinter? El widget Notebook en Tkinter es un contenedor que permite organizar múltiples páginas o pestañas en una sola ventana. Cada página en el Notebook se muestra como una pestaña y puede contener otros widgets y elementos de la interfaz gráfica.

El Widget Notebook es útil cuando se necesita presentar diferentes vistas o secciones en una sola ventana, como en aplicaciones con pestañas o pestañas de navegación. Los usuarios pueden cambiar entre las páginas haciendo clic en las pestañas correspondientes.

1.6 ¿Cómo implemento menús en Python con Tkinter? En Tkinter, se pueden implementar menús utilizando el widget Menu y agregándolo a la ventana principal. Aquí hay un ejemplo básico:

import tkinter as tk

def do\_something():

print("Doing something...")

root = tk.Tk()

# Crear una barra de menú

menubar = tk.Menu(root)

# Crear un menú desplegable

file\_menu = tk.Menu(menubar, tearoff=0)

file\_menu.add\_command(label="Open", command=do\_something)

file\_menu.add\_command(label="Save", command=do\_something)

file\_menu.add\_separator()

file\_menu.add\_command(label="Exit", command=root.quit)

# Agregar el menú desplegable a la barra de menú

menubar.add\_cascade(label="File", menu=file\_menu)

# Configurar la barra de menú en la ventana principal

root.config(menu=menubar)

root.mainloop()

En este ejemplo, se crea una ventana principal y se agrega una barra de menú mediante el widget Menu. Luego se crea un menú desplegable (File) con opciones como "Open", "Save" y "Exit", cada una asociada a una función. El menú desplegable se agrega a la barra de menú utilizando el método add\_cascade(). Finalmente, se configura la barra de menú en la ventana principal utilizando el atributo menu.

1.7 ¿Se puede realizar interfaces gráficas en Python con Paradigma estructurado y objetos? Sí, es posible realizar interfaces gráficas en Python utilizando tanto el paradigma estructurado como el paradigma orientado a objetos. La elección del paradigma depende de las necesidades y preferencias del desarrollador.

En el paradigma estructurado, la interfaz gráfica se construye definiendo funciones y procedimientos para manejar los eventos y la lógica de la interfaz. Se pueden utilizar estructuras de control y variables para gestionar los elementos visuales y las interacciones.

En el paradigma orientado a objetos, se crean clases que representan los elementos de la interfaz gráfica como objetos. Cada objeto puede tener propiedades y métodos que definen su comportamiento. Se pueden utilizar herencia y encapsulación para organizar y reutilizar el código relacionado con la interfaz.

1.8 ¿Describir los 3 métodos para colocar widgets en una GUI con Tkinter en Python? En Tkinter, hay tres métodos principales para colocar widgets en una GUI:

Pack(): Este método organiza los widgets en bloques horizontales o verticales. Los widgets se colocan uno después del otro y se ajustan automáticamente al tamaño del contenedor. Se utiliza el método pack() llamándolo en el widget que se desea colocar.

Grid(): Este método organiza los widgets en una cuadrícula de filas y columnas. Los widgets se colocan en celdas específicas de la cuadrícula utilizando coordenadas de fila y columna. Se utiliza el método grid() llamándolo en el widget y especificando los parámetros de fila y columna.

Place(): Este método permite colocar los widgets en posiciones absolutas utilizando coordenadas x e y. Los widgets se colocan en una ubicación específica dentro del contenedor utilizando el método place() llamándolo en el widget y especificando las coordenadas x e y.

1.9 ¿Cómo maneja los eventos en Tkinter? De ser posible ampliar en la web con "Bind". En Tkinter, los eventos se manejan utilizando el concepto de enlace (binding) de eventos. El enlace de eventos permite asociar una función o método a un evento específico generado por un widget. Cuando ocurre ese evento, se ejecuta la función o método asociado.

El método más común para enlazar eventos en Tkinter es el método bind(). Este método se llama en el widget al que se desea enlazar el evento y toma dos argumentos: el evento que se desea enlazar y la función o método que se ejecutará cuando ocurra el evento.

Aquí hay un ejemplo de enlace de evento utilizando el método bind():

import tkinter as tk

def handle\_click(event):

print("Button clicked!")

root = tk.Tk()

button = tk.Button(root, text="Click Me")

button.bind("<Button-1>", handle\_click)

button.pack()

root.mainloop()

En este ejemplo, se crea una ventana principal y un botón. Luego se utiliza el método bind() para enlazar el evento de clic del botón ("<Button-1>") a la función handle\_click(). Cuando se hace clic en el botón, se imprimirá "Button clicked!" en la consola.

Puedes ampliar tus conocimientos sobre la gestión de eventos en Tkinter consultando la documentación oficial de Tkinter o recursos en línea que explican en detalle cómo utilizar el método bind() y otros enfoques para manejar eventos en Tkinter

Python - SQLite

Marco Teórico:

1.1 ¿Qué es SQLite, cómo se integra a Python? SQLite es un sistema de gestión de bases de datos relacional ligero y de código abierto que se implementa como una biblioteca dentro de las aplicaciones. Proporciona una solución de base de datos de alto rendimiento y fácil de usar que no requiere un servidor separado. SQLite es compatible con la mayoría de los lenguajes de programación, incluido Python.

En Python, la integración con SQLite se logra a través del módulo sqlite3. Este módulo proporciona funciones y clases para interactuar con bases de datos SQLite desde Python. Permite la conexión a una base de datos, la ejecución de consultas SQL, la manipulación de datos y la gestión de transacciones.

1.2 ¿Cómo conecto a una base de datos SQLite desde Python? Para conectarte a una base de datos SQLite desde Python, debes seguir estos pasos:

Importar el módulo sqlite3: import sqlite3

Establecer una conexión con la base de datos utilizando la función connect(): connection = sqlite3.connect("nombre\_basedatos.db")

Crear un objeto cursor a partir de la conexión: cursor = connection.cursor()

Una vez que tienes la conexión establecida y el cursor creado, puedes ejecutar consultas SQL y manipular la base de datos utilizando métodos proporcionados por el cursor.

1.3 ¿Cómo ejecuto una sentencia SQL en Python + SQLite? Para ejecutar una sentencia SQL en Python + SQLite, debes seguir estos pasos:

Crear una conexión con la base de datos: connection = sqlite3.connect("nombre\_basedatos.db")

Crear un objeto cursor a partir de la conexión: cursor = connection.cursor()

Ejecutar la sentencia SQL utilizando el método execute(): cursor.execute("sentencia SQL")

Confirmar los cambios si es necesario: connection.commit()

La sentencia SQL puede ser cualquier instrucción válida en SQLite, como SELECT, INSERT, UPDATE, DELETE, CREATE TABLE, etc.

1.4 ¿Cómo creo una tabla en Python - SQLite? Para crear una tabla en Python - SQLite, debes ejecutar una sentencia SQL CREATE TABLE utilizando el método execute() del cursor. Aquí tienes un ejemplo:

import sqlite3

# Crear una conexión y un cursor

connection = sqlite3.connect("nombre\_basedatos.db")

cursor = connection.cursor()

# Ejecutar la sentencia SQL CREATE TABLE

cursor.execute("CREATE TABLE nombre\_tabla (columna1 TIPO, columna2 TIPO, ...)")

# Confirmar los cambios

connection.commit()

# Cerrar la conexión

connection.close()

En el ejemplo, debes reemplazar "nombre\_basedatos.db" por el nombre deseado para la base de datos y "nombre\_tabla" por el nombre deseado para la tabla. También debes especificar los nombres y tipos de columna adecuados.

1.5 Dar ejemplo de consultas (SELECT) que devuelvan todos los registros de una tabla. Para realizar una consulta SELECT que devuelva todos los registros de una tabla en Python - SQLite, debes seguir estos pasos:

import sqlite3

# Crear una conexión y un cursor

connection = sqlite3.connect("nombre\_basedatos.db")

cursor = connection.cursor()

# Ejecutar la consulta SELECT

cursor.execute("SELECT \* FROM nombre\_tabla")

# Obtener los resultados de la consulta

resultados = cursor.fetchall()

# Recorrer los resultados e imprimirlos

for fila in resultados:

print(fila)

# Cerrar la conexión

connection.close()

En el ejemplo, debes reemplazar "nombre\_basedatos.db" por el nombre de tu base de datos y "nombre\_tabla" por el nombre de la tabla deseada. La consulta SELECT se ejecuta con el método execute() del cursor, y los resultados se obtienen utilizando el método fetchall().

1.6 Dar ejemplo de consultas (SELECT) que devuelvan un registro de una tabla. Para realizar una consulta SELECT que devuelva un registro de una tabla en Python - SQLite, puedes utilizar el método fetchone() después de ejecutar la consulta. Aquí tienes un ejemplo:

import sqlite3

# Crear una conexión y un cursor

connection = sqlite3.connect("nombre\_basedatos.db")

cursor = connection.cursor()

# Ejecutar la consulta SELECT

cursor.execute("SELECT \* FROM nombre\_tabla WHERE columna = valor")

# Obtener el resultado de la consulta

registro = cursor.fetchone()

# Imprimir el registro

print(registro)

# Cerrar la conexión

connection.close()

En el ejemplo, debes reemplazar "nombre\_basedatos.db" por el nombre de tu base de datos, "nombre\_tabla" por el nombre de la tabla deseada, "columna" por el nombre de la columna en la que deseas buscar y "valor" por el valor que deseas buscar en esa columna.

1.7 Dar ejemplos de INSERT, DELETE y UPDATE en Python - SQLite. Aquí tienes ejemplos de cómo ejecutar las sentencias INSERT, DELETE y UPDATE en Python - SQLite:

Ejemplo de INSERT:

import sqlite3

# Crear una conexión y un cursor

connection = sqlite3.connect("nombre\_basedatos.db")

cursor = connection.cursor()

# Ejecutar la sentencia INSERT

cursor.execute("INSERT INTO nombre\_tabla (columna1, columna2, ...) VALUES (?, ?, ...)", (valor1, valor2, ...))

# Confirmar los cambios

connection.commit()

# Cerrar la conexión

connection.close()

En el ejemplo, debes reemplazar "nombre\_basedatos.db" por el nombre de tu base de datos, "nombre\_tabla" por el nombre de la tabla deseada, y proporcionar los valores correspondientes para las columnas en la sentencia INSERT.

Ejemplo de DELETE:

import sqlite3

# Crear una conexión y un cursor

connection = sqlite3.connect("nombre\_basedatos.db")

cursor = connection.cursor()

# Ejecutar la sentencia DELETE

cursor.execute("DELETE FROM nombre\_tabla WHERE columna = valor")

# Confirmar los cambios

connection.commit()

# Cerrar la conexión

connection.close()

En el ejemplo, debes reemplazar "nombre\_basedatos.db" por el nombre de tu base de datos, "nombre\_tabla" por el nombre de la tabla deseada, "columna" por el nombre de la columna en la que deseas realizar la eliminación y "valor" por el valor que deseas eliminar en esa columna.

Ejemplo de UPDATE:

import sqlite3

# Crear una conexión y un cursor

connection = sqlite3.connect("nombre\_basedatos.db")

cursor = connection.cursor()

# Ejecutar la sentencia UPDATE

cursor.execute("UPDATE nombre\_tabla SET columna1 = nuevo\_valor WHERE columna2 = valor")

# Confirmar los cambios

connection.commit()

# Cerrar la conexión

connection.close()

En el ejemplo, debes reemplazar "nombre\_basedatos.db" por el nombre de tu base de datos, "nombre\_tabla" por el nombre de la tabla deseada, "columna1" por el nombre de la columna que deseas actualizar, "nuevo\_valor" por el nuevo valor que deseas asignar y "columna2" por el nombre de la columna en la que deseas realizar la actualización y "valor" por el valor que deseas utilizar como condición.