

Bases de Datos Relacionales con Python

Guía rápida usando SQLite y Python 3.8

Esta guía hace uso de Visual Studio Code como editor de código. Los conceptos se aplican de forma general con el lenguaje Python 3.*

Antes de iniciar con el uso de la librería que permite gestionar una base de datos con comando SQL, repasemos conceptos de Python y el uso de la programación orientada por objetos en Python, de gran utilidad para el manejo de datos.

Clases en Python

Una clase en programación se puede considerar como la definición de un tipo de dato abstracto, que agrupa variables globales para las funciones que se agrupan bajo un nombre de un tipo de datos nuevo. Las variables globales en el nuevo tipo de datos la llamaremos **atributos** y a las funciones las llamaremos **métodos**.

Si deseas disponer de un tipo de datos nuevo llamado persona, el cual agrupa datos de nombre, peso y talla de una persona, y con esos datos deseas tener el índice de masa corporal, una solución es agrupar (encapsular) la función para el cálculo del índice de masa corporal, el cual emplea de las variables globales peso y talla para una persona.

Escrito en Python es:

clases.py

```
class Persona:
    def __init__(self, nombre, peso, talla):
        self.nombre = nombre
        self.peso = peso
        self.talla = talla

    def calcularIMC(self):
        valorIMC = self.peso / (self.talla * self.talla)
        return valorIMC
```

Python y SQLite

Guía rápida

La clase en Python se define con la creación de un grupo **class**

```
class nombreClase:
```

Recuerde los dos puntos **:** al final para indicar que inicia el grupo de instrucciones asociadas a la clase.

El método o función constructora es la que da los valores iniciales a los atributos (variables globales para la clase). Dado que Python no requiere definir el tipo de datos, es decir, no requiere la declaración de variables antes de usarlas e indicar el tipo de datos, la declaración de los atributos en una clase puede estar en el mismo método constructor.

En python el método constructor de una clase se declara con `__init__` (recuerda “`__`” son dos “`_`” seguidos), de esta forma:

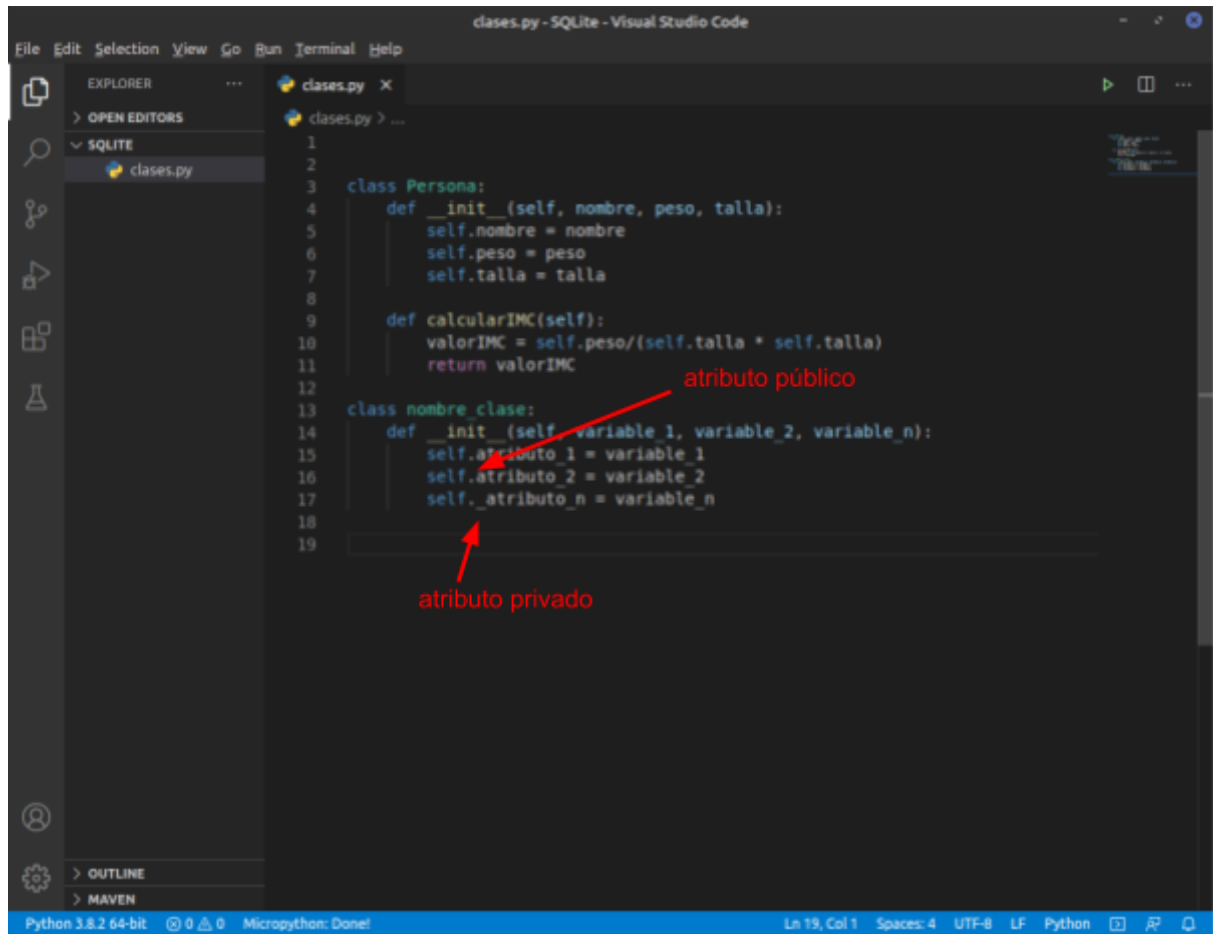
```
class nombre_clase:
    def __init__(self, variable_1, variable_2, variable_n):
        self.atributo_1 = variable_1
        self.atributo_2 = variable_2
        self.atributo_n = variable_n
```

Al anteponer `self.` a una variable, esta se convierte en atributos si está dentro del grupo de la clase.

Si deseo definir el atributo como privado, la instrucción es con un “`_`” antes del nombre. De esta forma: `self._atributo_1`

Python y SQLite

Guía rápida



De igual forma se puede asignar los métodos públicos y privados.

Un método es una función que está en el grupo de la clase (identendo) y en sus parámetros de entrada está el conjunto de atributos,bajo el nombre de self.

```
class nombre_clase:
    def __init__(self, variable_1, variable_2, variable_n):
        self.atributo_1 = variable_1
        self.atributo_2 = variable_2
        self._atributo_n = variable_n

    def metodo_1(self, otras_variables):
        variable_local_1 = otras_variables + self.atributo_1
```

De esta forma, se pueden usar los atributos con el inicio self. en la variable.

Todas las variables self. son atributos.

Python y SQLite

Guía rápida

Hay unas funciones que tiene las funciones estándar de los tipos de datos como:

Método	Función
Imprimir	<pre>def __str__(): # conformar un texto que visualiza los valores de # los atributos return texto</pre>
Comparar	<pre>def __eq__(valores_comparacion): #conformar el criterio de comparación return booleano (True o False) o entero</pre>

Tomemos como ejemplo, la propuesta de la clase persona para el cálculo del índice de masa corporal.

/clase.py

```
class Persona:  
    def __init__(self, nombre, peso, talla):  
        self.nombre = nombre  
        self.peso = peso  
        self.talla = talla  
  
    def calcularIMC(self):  
        valorIMC = self.peso/(self.talla * self.talla)  
        return valorIMC  
  
    def __str__(self):  
        texto = f'''  
Objeto de la Clase Persona:  
    Nombre: {self.nombre}  
    Peso: {self.peso}  
    Talla: {self.talla}  
    -----  
    IMC: {self.calcularIMC()}  
    '''  
        return texto
```

Python y SQLite

Guía rápida

Además, necesitamos un archivo de python extra que realiza el programa general.

/programa.py

```
# importamos el archivo python
# que contiene el código de la clase
from clases import *

#función de prueba del uso de objetos
def prueba_1():
    objeto_persona_1 = Persona('Edison', 96.3, 1.72)
    objeto_persona_2 = Persona('Juliany', 71.4, 1.63)

    print(objeto_persona_1)
    print(objeto_persona_2)

# se escribe la función principal para
# tener una buena práctica de diseño
def main():
    prueba_1()

if __name__ == '__main__':
    main()
```

En el archivo programa.py, el comando `objeto_persona_1 = Persona('Edison', 96.3, 1.72)` llama el método constructor cuando invoca `Persona(.....` y sus parámetros son almacenados en los atributos de la clase.

La instrucción `objeto_persona_1` en la instrucción `print`, llama el método `__str__` de la clase

Y el resultado es:

Comando: `python3 programa.py`

Resultado:

Objeto de la Clase Persona:

Nombre: Edison

Peso: 96.3

Talla: 1.72

IMC: 32.55137912385073

Python y SQLite

Guía rápida

Objeto de la Clase Persona:

Nombre: Juliany

Peso: 71.4

Talla: 1.63

IMC: 26.873423915089017

```
class Persona:
    def __init__(self, nombre, peso, talla):
        self.nombre = nombre
        self.peso = peso
        self.talla = talla

    def calcularIMC(self):
        valorIMC = self.peso/(self.talla * self.talla)
        return valorIMC

    def __str__(self):
        texto = f'''
Objeto de la Clase Persona:
Nombre: {self.nombre}
Peso: {self.peso}
Talla: {self.talla}
-----
IMC: {self.calcularIMC()}
'''
        return texto
```

```
# Importamos el archivo python
# que contiene el código de la clase
from clases import *

#función de prueba del uso de objetos
def prueba_1():
    objeto_persona_1 = Persona('Edison', 96.3, 1.72)
    objeto_persona_2 = Persona('Juliany', 71.4, 1.63)

    print(objeto_persona_1)
    print(objeto_persona_2)

# se escribe la función principal para
# tener una buena practica de diseño
def main():
    prueba_1()

if __name__ == '__main__':
    main()
```

```
edi@edi:~/Docencia/NoSQL-Python/SQLite$ python3 programa.py

Objeto de la Clase Persona:
Nombre: Edison
Peso: 96.3
Talla: 1.72
-----
IMC: 32.55137912385073

Objeto de la Clase Persona:
Nombre: Juliany
Peso: 71.4
Talla: 1.63
-----
IMC: 26.873423915089017
```

El código se puede descargar en:

- [clases.py](#)
- [programa.py](#)

Clase para gestionar SQLite

Practicemos el concepto de clases en Python definiendo una clase para la gestión de base de datos SQLite.

Declarar una clase para la gestión de bases de datos es útil, por:

- La conexión al ser un atributo, permanece activa por el tiempo de vida del objeto
- Bajo un mismo nombre de variable (instancia de una clase) se tienen las funciones del CRUD y otras relacionadas con la conexión de la base de datos
- En el método constructor se pueden verificar la existencia de tablas en la base datos o crearlas para garantizar la funcionalidad del programa

El código de la clase SQL es:

/capa_datos.py

```
import sqlite3 as sql

class SQL:
    def __init__(self, basedatos):
        self.db = sql.connect(basedatos+".db")
        self.cursor = self.db.cursor()
        self.ejecutar_ddl(comando_crear_tabla_producto)

    def cerrarConexion(self):
        try:
            self.db.close()
        except sql.OperationalError as errorSQL:
            return "Error en el programa: "+str(errorSQL)

#DDL: Lenguaje de definición de datos
# - Create: Crear estructura
# - Alter:  Modificar estructura
# - Drop:   Eliminar estructura
def ejecutar_ddl(self, comando_ddl):
    try:
        self.cursor.execute(comando_ddl)
    except sql.OperationalError as errorSQL:
        return "Error en el programa: "+str(errorSQL)
```

Python y SQLite

Guía rápida

```
#DML: Lenguaje de manipulación de datos
# - Select = seleccionar o buscar
# - Insert = añadir un registros
# - Update = actualizar un registro
# - Delete = borrar un regitro

def ejecutar_dml(self, comando_dml, datos=None):
    tipo = comando_dml.split()[0].upper()
    if tipo == 'SELECT':
        return self.ejecutar_select(comando_dml)
    if tipo == 'INSERT':
        return self.ejecutar_insert(comando_dml, datos)
    if tipo == 'UPDATE':
        return self.ejecutar_update(comando_dml, datos)
    if tipo == 'DELETE':
        return self.ejecutar_delete(comando_dml, datos)
    return 'Tipo de acción DML no conocida'

def ejecutar_select(self, comando):
    try:
        self.cursor.execute(comando)
        respuesta = self.cursor.fetchall()
        return respuesta
    except sql.OperationalError as errorSQL:
        return "Error en consulta: "+str(errorSQL)

def ejecutar_insert(self, comando_sql, datos):
    try:
        self.cursor.execute(comando_sql, datos)
        self.db.commit() #guardar los datos
        return "Operacion INSERT realizada"

    except sql.OperationalError as errorSQL:
        return "Error ejecutando comando: "+str(errorSQL)

def ejecutar_update(self, comando_sql, datos):
    try:
        self.cursor.execute(comando_sql,datos)
```


Python y SQLite

Guía rápida

```
self.db.commit()

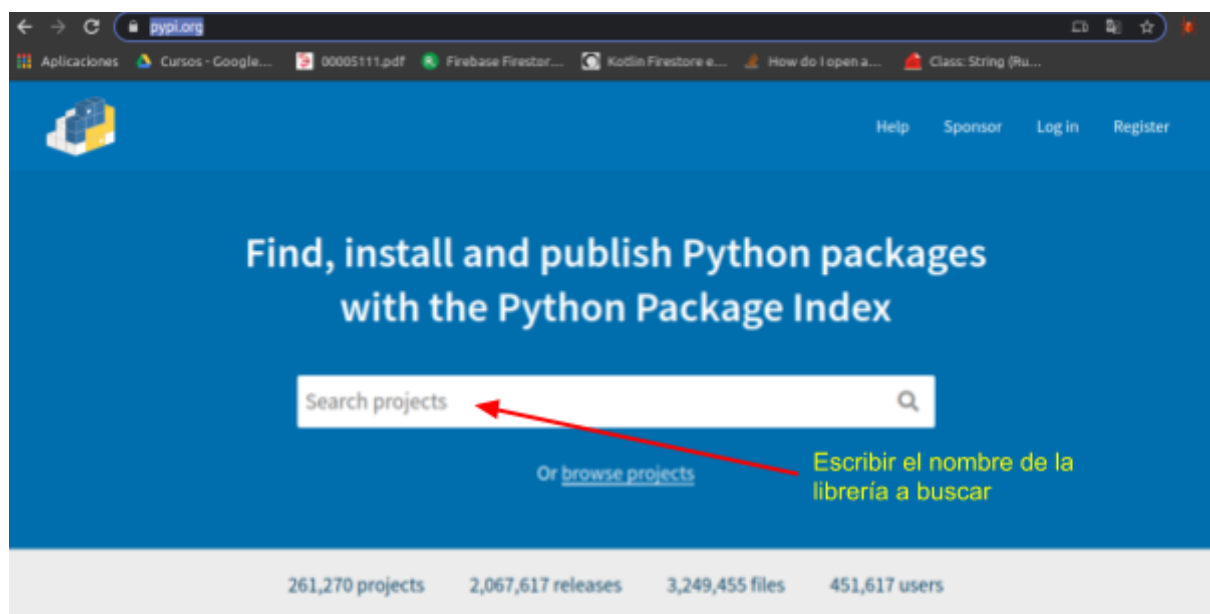
return "Actualización Ok."

except sql.OperationalError as errorSQL:
    return "Error en actualizar: " + str(errorSQL)

def ejecutar_delete(self, comando_sql, datos):
    try:
        self.cursor.execute(comando_sql,datos)
        self.db.commit()
        return "Eliminado con éxito."

    except sql.OperationalError as errorSQL:
        return "Error en actualizar: "+str(errorSQL)
```

Es importante instalar la librería SQLite. En la pagina pypi.org (<https://pypi.org/>) se encuentran muchas librerías útiles para python.



The Python Package Index (PyPI) is a repository of software for the Python programming language.

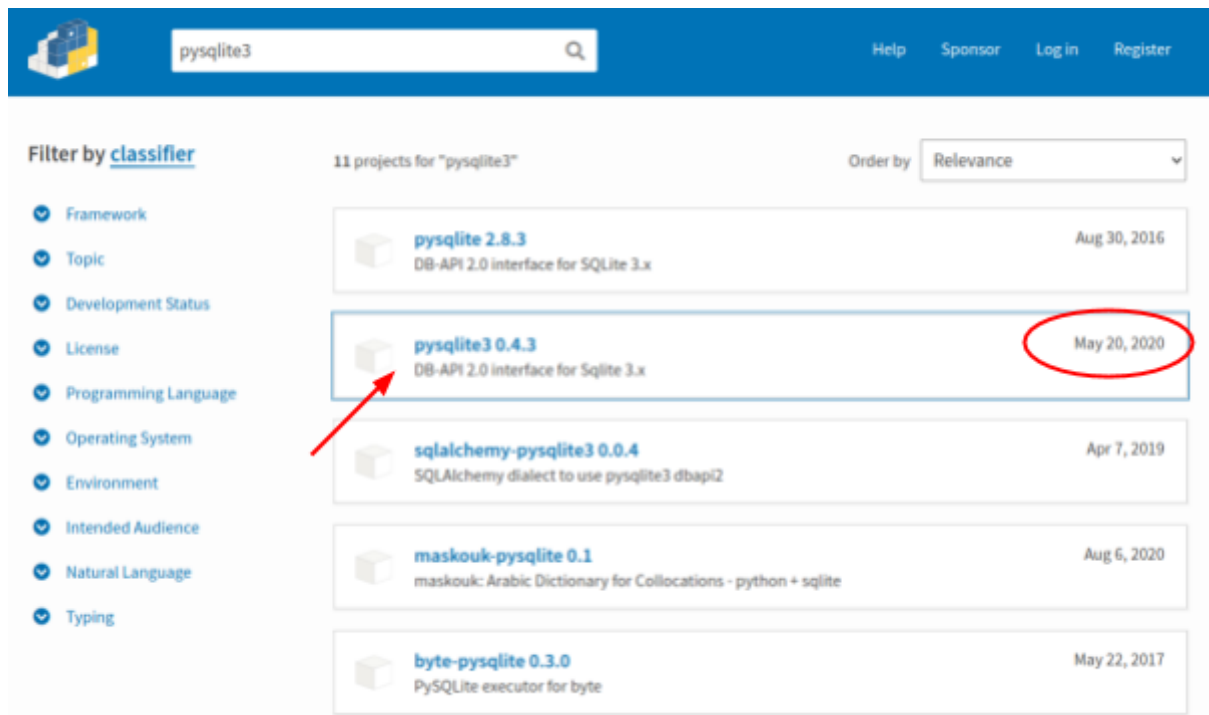
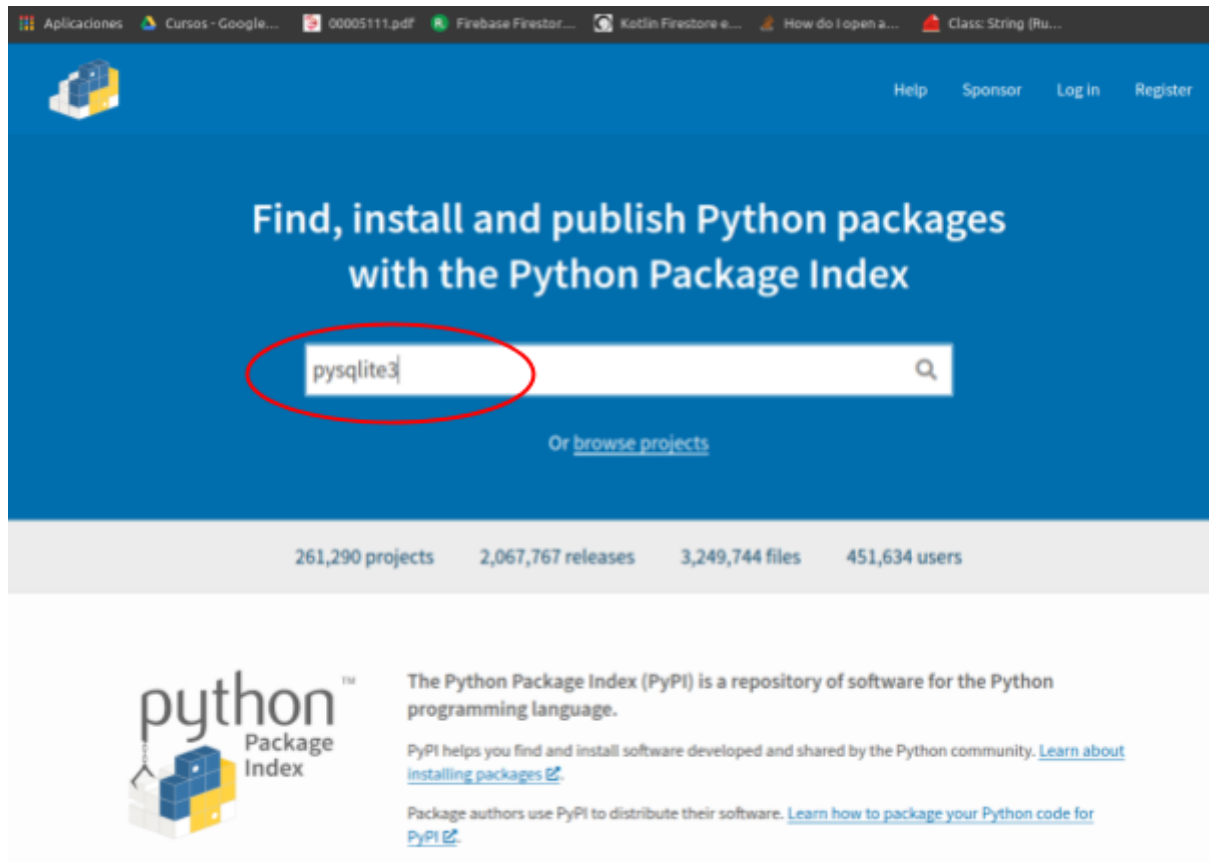
PyPI helps you find and install software developed and shared by the Python community. [Learn about installing packages](#).

Package authors use PyPI to distribute their software. [Learn how to package your Python code for PyPI](#).

Python y SQLite

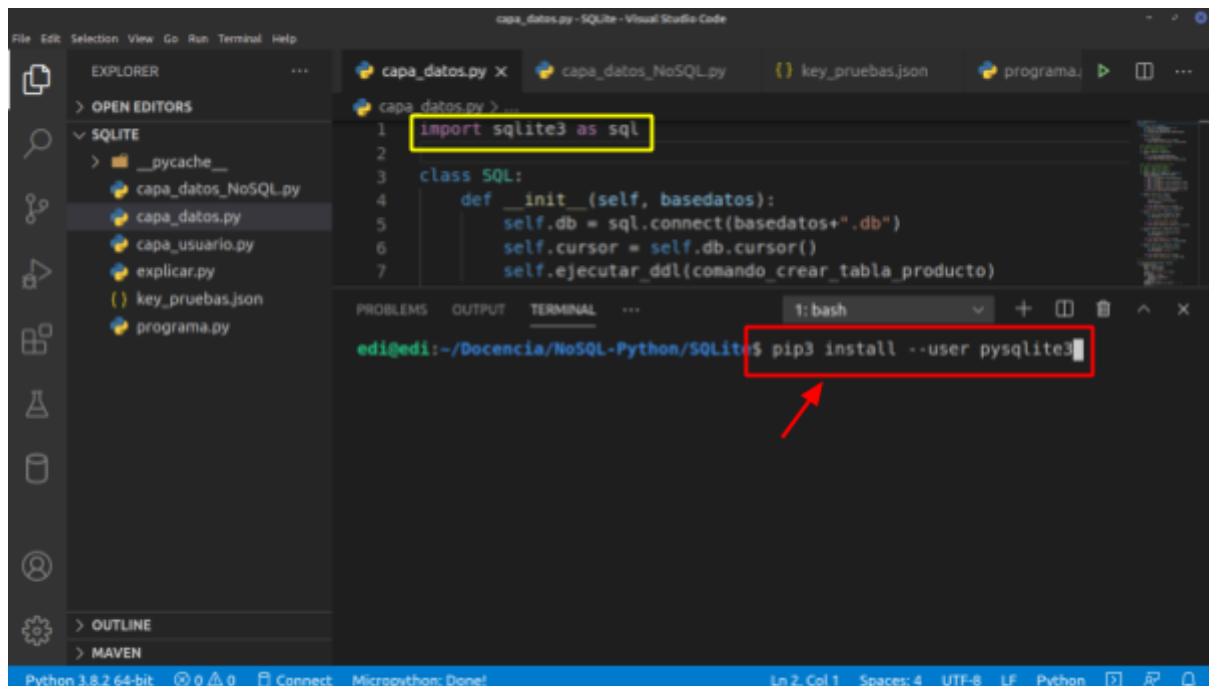
Guía rápida

Para SQLite, se debe instalar `pysqlite3`



Python y SQLite

Guía rápida



Se puede usar el comando de instalación PIP, con la instrucción **pip install --user pysqlite3** y en el código en python, se usa el comando de instalación: `import sqlite3 as sql`

Explicación de la Clase

Descomponemos la clase para entender cada método:

Método Constructor

```
def __init__(self, basedatos):  
    self.db = sql.connect(basedatos+".db")  
    self.cursor = self.db.cursor()  
    self.ejecutar_ddl(comando_crear_tabla_producto)
```

Recibe como parámetro el nombre de la base de datos que se desea crear.

Se crean dos atributos a la clase (tienen la palabras self. al inicio y son variables):

- **self.db**
Atributo que contiene el conector a la base de datos
- **self.cursor**
Atributo que relaciona el cursor para las operaciones en la base de datos

y se llama la función de crea la tabla con una instrucción tipo DDL (Lenguaje de Definición de Datos):

- **self.ejecutar_ddl()**
Convoca el método de la misma clase que ejecuta comandos tipo DDL.
El parámetro comando_crear_tabla_producto es un string que defina la creación de la tabla:

```
comando_crear_tabla_producto = """
CREATE TABLE IF NOT EXISTS productos(
    codigo TEXT NOT NULL,
    nombre TEXT NOT NULL,
    tipo TEXT NOT NULL,
    valor INTEGER NOT NULL );
"""
```

Método ejecutar_ddl

```
#DDL: Lenguaje de definición de datos
# - Create: Crear estructura
# - Alter:  Modificar estructura
# - Drop:   Eliminar estructura
def ejecutar_ddl(self, comando_ddl):
    try:
        self.cursor.execute(comando_ddl)
    except sql.OperationalError as errorSQL:
        return "Error en el programa: "+str(errorSQL)
```

El método ejecutar_ddl se orienta a realizar operaciones para la creación o eliminación de estructura de datos en la base de datos.

El método recibe una instrucción escrita en lenguaje SQL, la instrucción se almacena en la variable comando_ddl.

Python y SQLite

Guía rápida

La instrucción **self.cursor.execute** hace que el atributo cursor, el cual tiene la conexión a la base de datos, ejecute “execute” la instrucción escrita en comando_ddl.

Este método tiene un manejo de excepciones (“errores”) con los operadores **try** y **except**. Todas las instrucciones que se escriben en el bloque de instrucciones try, se observan si presentan error de ejecución o no. Cuando se presenta un error, se ejecuta el bloque de instrucciones que están except.

Método ejecutar_dml

```
#DML: Lenguaje de manipulación de datos
# - Select = seleccionar o buscar
# - Insert = añadir un registros
# - Update = actualizar un registro
# - Delete = borrar un regitro
def ejecutar_dml(self, comando_dml, datos=None):
    tipo = comando_dml.split()[0].upper()
    if tipo == 'SELECT':
        return self.ejecutar_select(comando_dml)
    if tipo == 'INSERT':
        return self.ejecutar_insert(comando_dml, datos)
    if tipo == 'UPDATE':
        return self.ejecutar_update(comando_dml, datos)
    if tipo == 'DELETE':
        return self.ejecutar_delete(comando_dml, datos)
    return 'Tipo de acción DML no conocida'
```

El método ejecutar_dml se enfoca en atender la manipulación de datos en la base de datos, con instrucciones CRUD.

Antes de ejecutar los comandos en la base de datos, se hace un verificación de cantidad de parametros según operación, es decir, para las operaciones de insertar, actualizar y borrar, se debe enviar los datos, y por lo general regresan un mensaje de confirmación o rechazo. La operación seleccionar por el contrario, no envía datos pero recupera un conjunto de valores en una lista.

Este método toma la primera palabra de la instrucción e identifica si es SELECT, INSERT, UPDATE o DELETE, y llama el método respectivo.

Método ejecutar_select

```
def ejecutar_select(self, comando):
    try:
        self.cursor.execute(comando)

        respuesta = self.cursor.fetchall()
        return respuesta

    except sql.OperationalError as errorSQL:
        return "Error en consulta: "+str(errorSQL)
```

El método ejecutar_select se diferencia de los otros metodos insert, update y delete, porque recupera un conjunto de datos con la instrucción self.cursor.fetchall() . La instrucción fetchall adquiere todos los resultados al ejecutar el comando execute, entregando una lista

Este método no realiza la instrucción commit, porque no envía datos.

Método ejecutar_insert

```
def ejecutar_insert(self, comando_sql, datos):
    try:
        self.cursor.execute(comando_sql, datos)
        self.db.commit() #guardar los datos
        return "Operacion INSERT realizada"

    except sql.OperationalError as errorSQL:
        return "Error ejecutando comando: "+str(errorSQL)
```

Los métodos ejecutar_insert, ejecutar_update y ejecutar_delete, son iguales. Se que el código puede estar mejor escrito :(La razón es dar claridad.

Método ejecutar_update

```
def ejecutar_update(self, comando_sql, datos):
    try:
        self.cursor.execute(comando_sql,datos)
        self.db.commit()
        return "Actualización Ok."
```

```
except sql.OperationalError as errorSQL:
    return "Error en actualizar: " + str(errorSQL)
```

Método ejecutar_delete

```
def ejecutar_delete(self, comando_sql, datos):
    try:
        self.cursor.execute(comando_sql, datos)
        self.db.commit()
        return "Eliminado con éxito."

    except sql.OperationalError as errorSQL:
        return "Error en actualizar: "+str(errorSQL)
```

Y para control en la creación de comandos, se han adicionado dos funciones:

Función crearSQLcomandoInsert

```
def crearSQLcomandoInsert(tabla, valores):
    if valores != None:
        datos = valores.values()
        campos = valores.keys()

        comando_sql = f"INSERT INTO {tabla} ("
        parametros = '('
        for etiqueta in campos:
            comando_sql += etiqueta + ','
            parametros += '?,'
        comando_sql += ')'
        comando_sql = comando_sql.replace(',)', ')')
        parametros += ')'
        parametros = parametros.replace(',)', ')')
        comando_sql += ' VALUES ' + parametros

        return {'comandoSQL':comando_sql, 'campos':list(campos),
            'datos':list(datos)}
```

Python y SQLite

Guía rápida

```
else:  
    return None
```

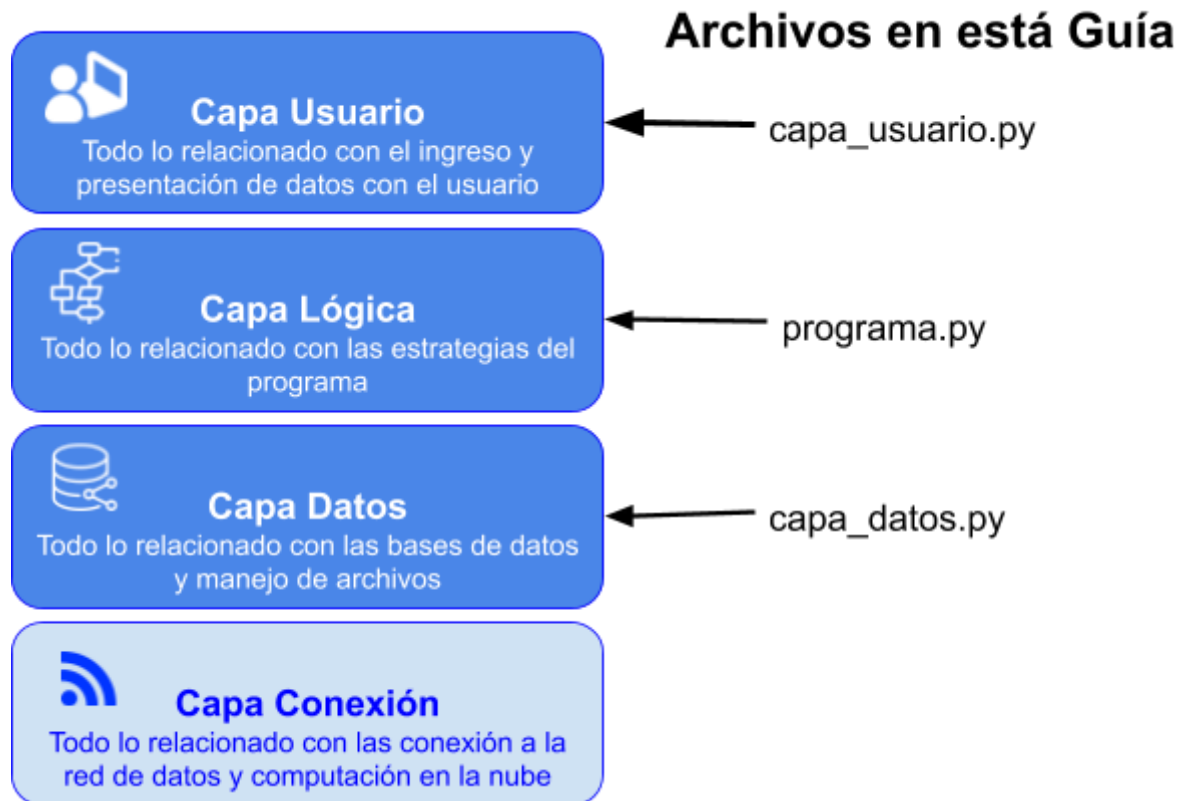
Función crearSQLcomandoUpdate

```
def crearSQLcomandoUpdate(tabla, valores, rowid):  
    if valores != None:  
        datos = list(valores.values())  
        campos = list(valores.keys())  
  
        comando_sql = f"UPDATE {tabla} SET "  
        for etiqueta in campos:  
            comando_sql += etiqueta + '=?,'  
        comando_sql += ')'  
        comando_sql = comando_sql.replace(',)', ' WHERE rowid=?;')  
        datos.append(rowid)  
        return {'comandoSQL':comando_sql, 'campos':list(campos),  
            'datos':list(datos)}  
    else:  
        return None
```

Ambas funciones reciben el conjunto de datos y crean una instrucción válida en SQL para insert y actualizar los datos.

Hay que observar que no son métodos, son funciones al estar fuera de la clase y no tener el parámetro de entrada self.

PROGRAMAS EN TRES CAPAS



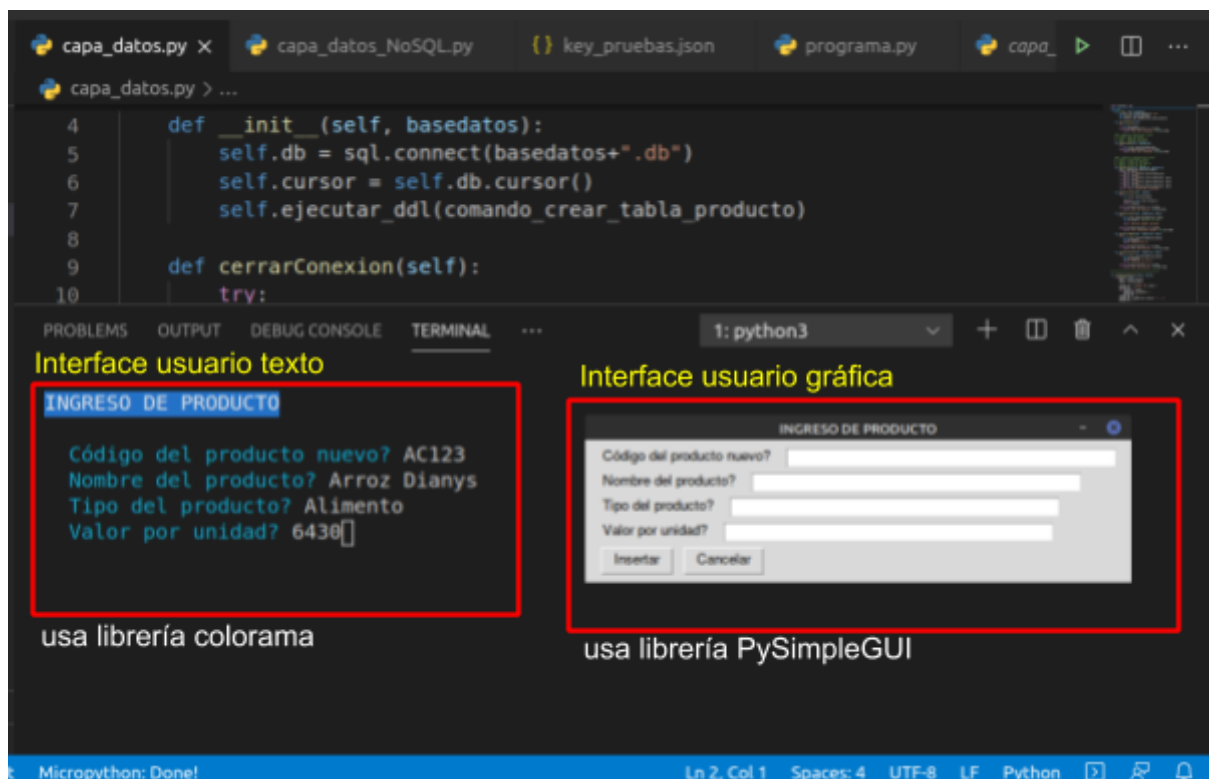
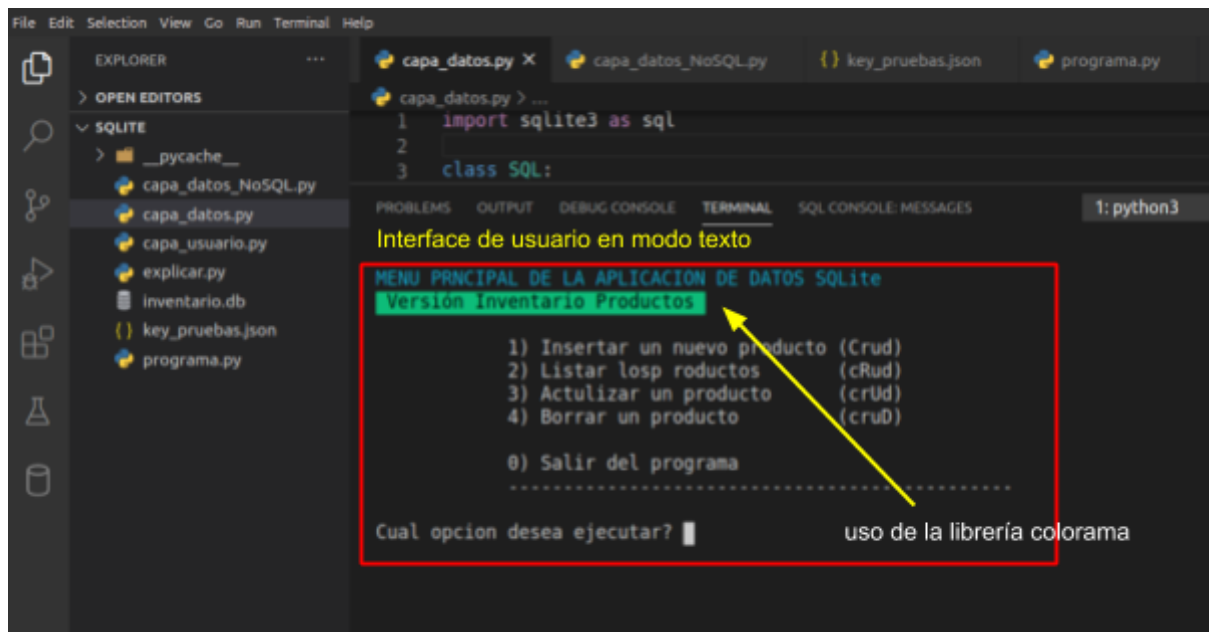
Archivos a descargar:

- [capa_usuario.py](#)
 - [programa.py](#)
 - [capa_datos.py](#)
-

Python y SQLite

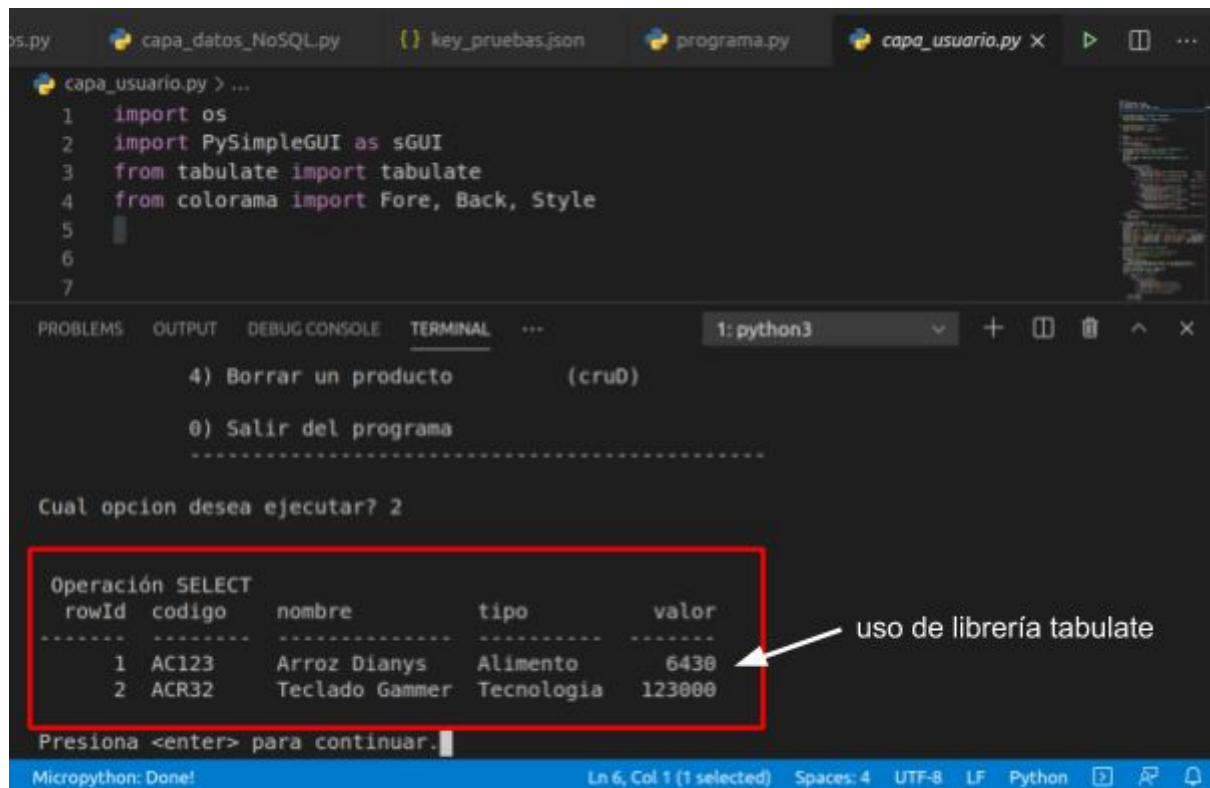
Guía rápida

Capa usuario



Python y SQLite

Guía rápida



```
capa_usuario.py > ...
1 import os
2 import PySimpleGUI as sGUI
3 from tabulate import tabulate
4 from colorama import Fore, Back, Style
5
6
7
```

4) Borrar un producto (cruD)

0) Salir del programa

Cual opción desea ejecutar? 2

Operación	SELECT			
rowId	codigo	nombre	tipo	valor
1	AC123	Arroz Dianys	Alimento	6430
2	ACR32	Teclado Gammer	Tecnologia	123000

Presiona <enter> para continuar.

uso de librería tabulate

El código para la capa de usuario se basa en la definición de funciones para los usos específicos. No se crea una clase porque no se deben conservar los datos que se muestran o se capta.

Recuerde que una vez la función termina, sus variables desaparecen de la memoria.

/capa_usuario.py

```
import os
import PySimpleGUI as sGUI
from tabulate import tabulate
from colorama import Fore, Back, Style

def verTablaDatos(titulo, etiquetas, respuestas):
    print('\n\n', titulo)
    print(tabulate(respuestas, headers=etiquetas), '\n')

def verMensajeDatos(titulo, mensaje):
    print('\n\n', titulo)
    print(' Respuesta >> ', mensaje, '\n')

def stop():
```

Python y SQLite

Guía rápida

```
input("Presiona <enter> para continuar.")

def captarInt(mensaje):
    return int(input(mensaje))

def captarDatosProductoTXTx(titulo, estructura, datoAct=None):
    os.system('cls') #windows limpiar consolo
    os.system('clear') #linux y MacOS limpiar consola
    print()
    print(Fore.WHITE + Back.BLUE + titulo + Style.RESET_ALL + '\n')
    valores = {}
    try:
        i=1
        for item in estructura:
            if datoAct == None:
                if item['tipo']=='int':
                    valores[item['nombre']]=int(input(Fore.CYAN + ' ' +
+ item['texto'] + Style.RESET_ALL))
                elif item['tipo']=='float':
                    valores[item['nombre']]=float(input(Fore.CYAN + ' ' +
+ item['texto'] + Style.RESET_ALL))
                else:
                    valores[item['nombre']]=input(Fore.CYAN + ' ' +
+ item['texto'] + Style.RESET_ALL)
            else:
                if item['tipo']=='int':
                    valores[item['nombre']]=input(Fore.CYAN + ' ' +
+ item['texto']+' ('+ str(datoAct[i])+') ' + Style.RESET_ALL)
                    if valores[item['nombre']] == '':
                        valores[item['nombre']] = int(datoAct[i])
                elif item['tipo']=='float':
                    valores[item['nombre']]=input(Fore.CYAN + ' ' +
+ item['texto']+' ('+ str(datoAct[i])+') ' + Style.RESET_ALL)
                    if valores[item['nombre']] == '':
                        valores[item['nombre']] = float(datoAct[i])
                else:
                    valores[item['nombre']]=input(Fore.CYAN + ' ' +
+ item['texto'] + ' ('+ datoAct[i] + ') ' + Style.RESET_ALL)
                    if valores[item['nombre']] == '':
```

Python y SQLite

Guía rápida

```
        valores[item['nombre']] = str(datoAct[i])

        i+=1

    return valores

except:
    mensajeError('Error en tipo de datos','Por favor verifique el
tipo de dato e intente de nuevo.')
    return None

def captarDatosProductoTXT():
    os.system('cls') #windows limpiar consolo
    os.system('clear') #linux y MacOS limpiar consola
    print()
    print(Fore.WHITE + Back.BLUE + "INGRESO DE PRODUCTOS" +
Style.RESET_ALL + '\n')
    valores = {}
    valores['codigo'] = input(Fore.CYAN + " Código del producto? " +
Style.RESET_ALL)
    valores['nombre'] = input(Fore.CYAN + " Nombre del producto? " +
Style.RESET_ALL)
    valores['tipo'] = input(Fore.CYAN + " Tipo del producto? " +
Style.RESET_ALL)
    valores['valor'] = input(Fore.CYAN + " Valor por unidad? " +
Style.RESET_ALL)
    return valores

def captarDatosProductoGUI(titulo, estructura):
    valores={}

    #print(sGUI.theme_list()) #Ver los temas disponibles
    sGUI.theme('SystemDefault1') # tema del color
    #Crea los componentes del a ventana
    componentes = []
    for item in estructura:

componentes.append([sGUI.Text(item['texto']),sGUI.InputText()])
        componentes.append([sGUI.Button('Insertar'),
sGUI.Button('Cancelar')])
        # Crear la ventana
        window = sGUI.Window(titulo, componentes)
        eventoClick, values = window.read()
```

Python y SQLite

Guía rápida

```
try:
    if eventoClick == 'Insertar':
        i = 0
        for item in estructura:
            if item['tipo']=='int':
                valores[item['nombre']]=int(values[i])
            elif item['tipo']=='float':
                valores[item['nombre']]=float(values[i])
            else:
                valores[item['nombre']]=values[i]
            i+=1
        window.close()
        return valores
except:
    window.close()
    mensajeError('Error en tipo de datos','Por favor verifique el
tipo de dato e intente de nuevo.')
    return None

def menu_principal():
    while True:
        os.system('cls') #windows limpiar consolo
        os.system('clear') #linux y MacOS limpiar consola
        print()
        print(Fore.CYAN + "MENU PRNCIPAL DE LA APLICACIÓN DE DATOS
SQLite" + Style.RESET_ALL)
        print(Fore.BLACK + Back.GREEN + " Versión Inventario Productos
" + Style.RESET_ALL)
        print(
            '''
            1) Insertar un nuevo producto (Crud)
            2) Listar los productos (cRud)
            3) Actualizar un producto (crUd)
            4) Borrar un producto (cruD)

            0) Salir del programa
            -----
            ''')
```

Python y SQLite

Guía rápida

```
)  
opcion = int(input('Cual opcion desea ejecutar? '))  
if opcion >= 0 and opcion <= 4:  
    return opcion  
mensajeError('ERROR EN LA OPCIÓN','Opción no valida,  
seleccione otra opción')  
  
def mensajeError(titulo, mensaje):  
    # Colorama  
    # Fore: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE,  
    RESET.  
    # Back: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE,  
    RESET.  
    # Style: DIM, NORMAL, BRIGHT, RESET_ALL  
    print()  
    print(Fore.RED + Back.YELLOW + titulo + Style.RESET_ALL)  
    print(Fore.RED + mensaje + Style.RESET_ALL)  
    input("Presiona <enter> para continuar.")
```

Las funciones por separado:

Librerías

```
import os  
import PySimpleGUI as sGUI  
from tabulate import tabulate  
from colorama import Fore, Back, Style
```

- **os:**
Librería del sistema operativo. No requiere instalación
Se emplea en el código para limpiar la pantalla en texto
- **PySimpleGUI:**
Librería para la gestión de ventanas graficas.
Se instala con pip **pip install pysimplegui**
Se emplea en el código para el ingreso de dato
- **tabular:**
Libería para la visualización de listas, tuplas y diccionarios con las tabulaciones por

Python y SQLite

Guía rápida

columnas

Se instala con pip `pip install tabulate`

Se emplea en el código para listar los datos de la base de datos

- colorama:

Librería para pintar el texto en la terminal

Se instala con pip `pip install colorama`

Se emplea en el código para el menú, mensaje de error y el ingreso de datos

Función verTablaDatos

```
def verTablaDatos(titulo, etiquetas, respuestas):  
    print('\n\n', titulo)  
    print(tabulate(respuestas, headers=etiquetas), '\n')
```

Esta función muestra la lista de etiquetas que son los nombres de las columnas, que está definido en la `capa_datos`, y muestra la lista de datos que es la respuesta de instrucciones como `Select`.

Función verMensajeDatos

```
def verMensajeDatos(titulo, mensaje):  
    print('\n\n', titulo)  
    print(' Respuesta >> ', mensaje, '\n')
```

Esta función muestra un mensaje general y lo relaciona con un título

Función mensajeError

```
def mensajeError(titulo, mensaje):  
    # Colorama  
    # Fore: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE,  
    RESET.  
    # Back: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE,  
    RESET.  
    # Style: DIM, NORMAL, BRIGHT, RESET_ALL  
    print()  
    print(Fore.RED + Back.YELLOW + titulo + Style.RESET_ALL)  
    print(Fore.RED + mensaje + Style.RESET_ALL)  
    input("Presiona <enter> para continuar.")
```

Función para mostrar un error llamativo, con color rojo.

Además espera a ser leído, con la instrucción `input` sin asignación a variable

Función stop

```
def stop():  
    input("Presiona <enter> para continuar.")
```

Función que detiene el despliegue de la pantalla hasta que se presione la tecla enter.

Función captarInt

```
def captarInt(mensaje):  
    return int(input(mensaje))
```

Función para captar un valor entero. Utilizado para escribir el rowId del conjunto de datos

Función captarDatosProductoTXTx

```
def captarDatosProductoTXTx(titulo, estructura, datoAct=None):  
    os.system('cls') #windows limpiar consola  
    os.system('clear') #linux y MacOS limpiar consola  
    print()  
    print(Fore.WHITE + Back.BLUE + titulo + Style.RESET_ALL + '\n')  
    valores = {}  
    try:  
        i=1  
        for item in estructura:  
            if datoAct == None:  
                if item['tipo']=='int':  
                    valores[item['nombre']] = int(input(Fore.CYAN + ' ' +  
+ item['texto'] + Style.RESET_ALL))  
                elif item['tipo']=='float':  
                    valores[item['nombre']] = float(input(Fore.CYAN + ' '  
+ item['texto'] + Style.RESET_ALL))  
                else:  
                    valores[item['nombre']] = input(Fore.CYAN + ' ' +  
item['texto'] + Style.RESET_ALL)  
            else:  
                if item['tipo']=='int':  
                    valores[item['nombre']] = input(Fore.CYAN + ' ' +  
item['texto']+' ('+ str(datoAct[i])+') ' + Style.RESET_ALL)  
                    if valores[item['nombre']] == '':  
                        valores[item['nombre']] = int(datoAct[i])
```

Python y SQLite

Guía rápida

```
        elif item['tipo']=='float':
            valores[item['nombre']]=input(Fore.CYAN + ' ' +
item['texto']+' ('+ str(datoAct[i])+') ' + Style.RESET_ALL)
            if valores[item['nombre']] == '':
                valores[item['nombre']] = float(datoAct[i])
        else:
            valores[item['nombre']]=input(Fore.CYAN + ' ' +
item['texto'] + ' ('+ datoAct[i])+') ' + Style.RESET_ALL)
            if valores[item['nombre']] == '':
                valores[item['nombre']] = str(datoAct[i])

        i+=1

    return valores
except:
    mensajeError('Error en tipo de datos','Por favor verifique el
tipo de dato e intente de nuevo.')
    return None
```

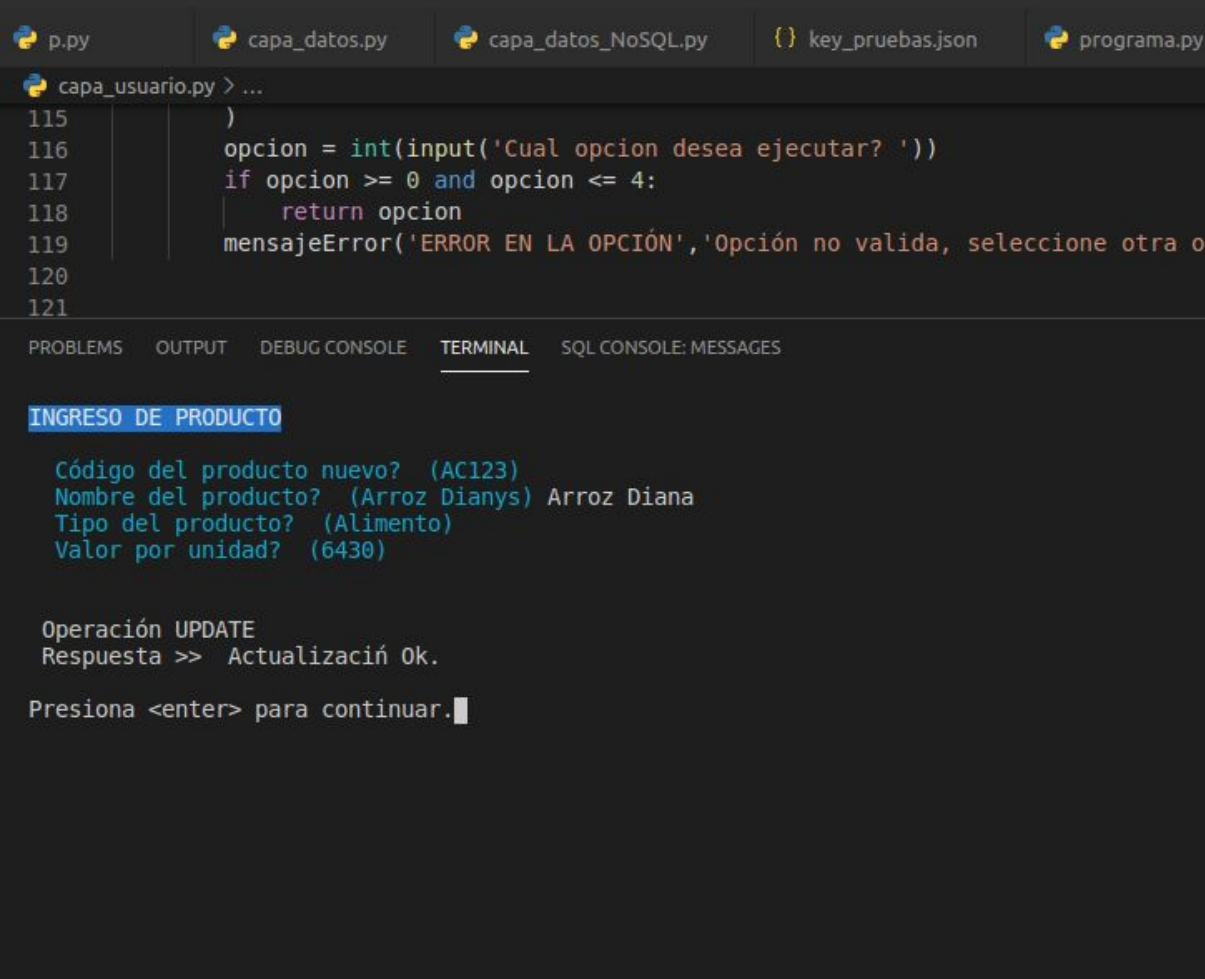
Función para captar los datos (“formulario”) en modo texto que genera los campos según la estructura.

```
estructura_producto = [
    {'nombre':'codigo', 'texto':'Código del producto nuevo? ', 'tipo':'string'},
    {'nombre':'nombre', 'texto':'Nombre del producto? ', 'tipo':'string'},
    {'nombre':'tipo', 'texto':'Tipo del producto? ', 'tipo':'string'},
    {'nombre':'valor', 'texto':'Valor por unidad? ', 'tipo':'int'}
]
```

El parámetro de entrada datoAct es para mostrar datos que ya están en el registro. Útil para actualizar valores, presentado entre paréntesis los valores y permite programa, presionar enter si el valor es igual, o escribir el nuevo valor.

Python y SQLite

Guía rápida



The screenshot shows a Python IDE with several tabs: p.py, capa_datos.py, capa_datos_NoSQL.py, key_pruebas.json, and programa.py. The active file is capa_usuario.py, showing lines 115 to 121. The code defines a function to handle menu options. The terminal window at the bottom shows the execution of the program, including the 'INGRESO DE PRODUCTO' section where user input is captured and an UPDATE operation is performed.

```
capa_usuario.py > ...
115         )
116         opcion = int(input('Cual opcion desea ejecutar? '))
117         if opcion >= 0 and opcion <= 4:
118             return opcion
119         mensajeError('ERROR EN LA OPCIÓN','Opción no valida, seleccione otra o
120
121
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE: MESSAGES

INGRESO DE PRODUCTO

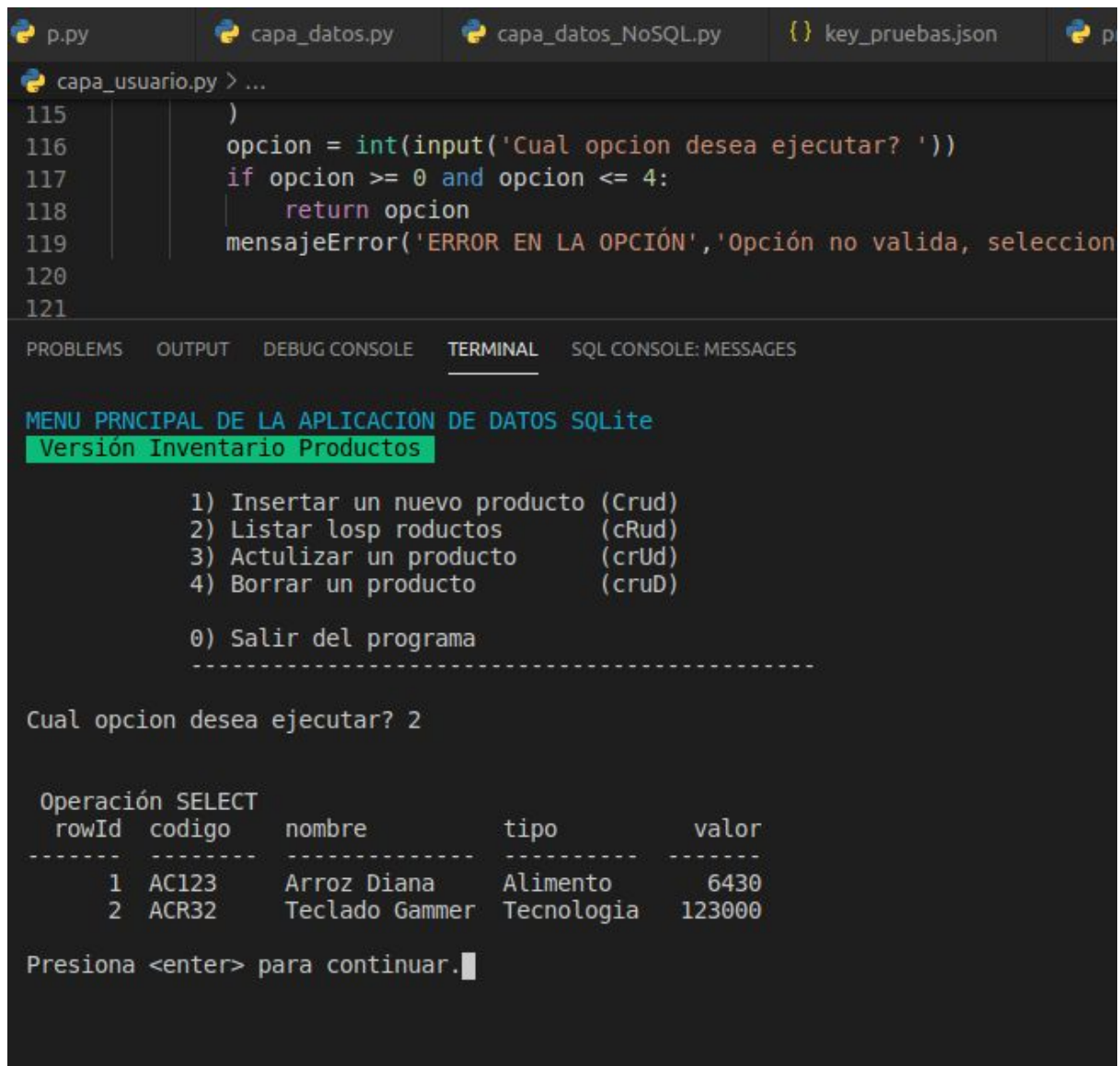
Código del producto nuevo? (AC123)
Nombre del producto? (Arroz Dianys) Arroz Diana
Tipo del producto? (Alimento)
Valor por unidad? (6430)

Operación UPDATE
Respuesta >> Actualizaciñ Ok.

Presiona <enter> para continuar.█

Python y SQLite

Guía rápida



```
capa_usuario.py > ...
115         )
116         opcion = int(input('Cual opcion desea ejecutar? '))
117         if opcion >= 0 and opcion <= 4:
118             return opcion
119         mensajeError('ERROR EN LA OPCIÓN','Opción no valida, seleccion
120
121
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** SQL CONSOLE: MESSAGES

MENU PRINCIPAL DE LA APLICACION DE DATOS SQLite
Versión Inventario Productos

- 1) Insertar un nuevo producto (Crud)
- 2) Listar los productos (cRud)
- 3) Actualizar un producto (crUd)
- 4) Borrar un producto (cruD)
- 0) Salir del programa

Cual opcion desea ejecutar? 2

Operación SELECT

rowId	codigo	nombre	tipo	valor
1	AC123	Arroz Diana	Alimento	6430
2	ACR32	Teclado Gammer	Tecnologia	123000

Presiona <enter> para continuar.

Función captarDatosProductoTXT

```
def captarDatosProductoTXT():
    os.system('cls') #windows limpiar consolo
    os.system('clear') #linux y MacOS limpiar consola
    print()
    print(Fore.WHITE + Back.BLUE + "INGRESO DE PRODUCTOS" +
    Style.RESET_ALL + '\n')
    valores = {}
    valores['codigo'] = input(Fore.CYAN + " Código del producto? " +
    Style.RESET_ALL)
    valores['nombre'] = input(Fore.CYAN + " Nombre del producto? " +
    Style.RESET_ALL)
    valores['tipo'] = input(Fore.CYAN + " Tipo del producto? " +
```

Python y SQLite

Guía rápida

```
Style.RESET_ALL)
    valores['valor'] = input(Fore.CYAN + " Valor por unidad? " +
Style.RESET_ALL)
    return valores
```

Función para formulario texto fijo

la instrucción `print(Fore.WHITE + Back.BLUE + "INGRESO DE PRODUCTOS" + Style.RESET_ALL + '\n')` `Fore.WHITE` activa la letra en blanco y `Back.BLUE` activa el fondo azul en la letra blanca. Para regresar a la normalidad el texto, se adiciona la instrucción `Style.RESET_ALL`.

Función captarDatosProductoGUI

```
def captarDatosProductoGUI(titulo, estructura):
    valores={}
    #print(sGUI.theme_list()) #Ver los temas disponibles
    sGUI.theme('SystemDefault1') # tema del color
    #Crea los componentes de la ventana
    componentes = []
    for item in estructura:

componentes.append([sGUI.Text(item['texto']),sGUI.InputText()])
        componentes.append([sGUI.Button('Insertar'),
sGUI.Button('Cancelar')])
    # Crear la ventana
    window = sGUI.Window(titulo, componentes)
    eventoClick, values = window.read()
    try:
        if eventoClick == 'Insertar':
            i = 0
            for item in estructura:
                if item['tipo']=='int':
                    valores[item['nombre']]=int(values[i])
                elif item['tipo']=='float':
                    valores[item['nombre']]=float(values[i])
                else:
                    valores[item['nombre']]=values[i]
```

Python y SQLite

Guía rápida

```
        i+=1
    window.close()
    return valores
except:
    window.close()
    mensajeError('Error en tipo de datos','Por favor verifique el
tipo de dato e intente de nuevo.')
    return None
```

Función “formulario” en GUI - Interfaz Grafica de Usuario- empleando la librería PySimpleGUI.

Todo es gestionado como una lista, `[sGUI.Text(item['texto']),sGUI.InputText()]` haciendo una interface de forma realmente simple. Igual es el envío de los campos, se crea una lista `eventoClick, values = window.read()`, en **values**, que es una lista, se tienen todos los valores escritos en los campos de la interfaz gráfica.

Función

```
def menu_principal():
    while True:
        os.system('cls') #windows limpiar consolo
        os.system('clear') #linux y MacOS limpiar consola
        print()
        print(Fore.CYAN + "MENU PRNCIPAL DE LA APLICACIÓN DE DATOS
SQLite" + Style.RESET_ALL)
        print(Fore.BLACK + Back.GREEN + " Versión Inventario Productos
" + Style.RESET_ALL)
        print(
            '''
            1) Insertar un nuevo producto (Crud)
            2) Listar los productos          (cRud)
            3) Actualizar un producto        (crUd)
            4) Borrar un producto            (cruD)

            0) Salir del programa
            -----
            '''
        )
        opcion = int(input('Cual opcion desea ejecutar? '))
```

Python y SQLite

Guía rápida

```
if opcion >= 0 and opcion <= 4:
    return opcion
mensajeError('ERROR EN LA OPCIÓN','Opción no valida,
seleccione otra opción')
```

Y la capa de lógica, es el programa:

prorgama.py

```
from capa_datos import *
from capa_usuario import *

def main():
    #coenxion
    nombreBaseDatos = 'inventario'
    objDB = SQL(nombreBaseDatos)
    while True:
        opcion = menu_principal()
        if opcion == 0:
            break
        elif opcion == 1: #Insertar
            datosIngreso = captarDatosProductoGUI(titulo_producto,
estructura_producto)
            #datosIngreso = captarDatosProductoTXTx(titulo_producto,
estructura_producto)
            if datosIngreso != None:
                comando =
crearSQLcomandoInsert('productos',datosIngreso)
                respuesta_sql =
objDB.ejecutar_dml(comando['comandoSQL'],comando['datos'])
                verMensajeDatos('Operación INSERT', respuesta_sql)
                stop()
            else:
                verMensajeDatos('Operación INSERT', "Erorr en la
ingrese de datos por el usuario")
                stop()
            pass
        elif opcion == 2: #Consultar
            comando_sql = "SELECT rowid,* FROM productos;"
```

Python y SQLite

Guía rápida

```
        respuesta_sql = objDB.ejecutar_dml(comando_sql)
        verTablaDatos('Operación SELECT', etiquetas_producto,
respuesta_sql)
        stop()
        pass
    elif opcion == 3: #Actualizar
        comando_sql = "SELECT rowid,* FROM productos;"
        respuesta_sql = objDB.ejecutar_dml(comando_sql)
        verTablaDatos('Resgistros Activos:', etiquetas_producto,
respuesta_sql)
        idReg = captarInt('Cual número de registro desea
Actualizar? ')
        comando = crearSQLcomandoUpdate('productos',
captarDatosProductoTXTx(titulo_producto, estructura_producto,
respuesta_sql[idReg-1]),idReg)
        respuesta_sql =
objDB.ejecutar_dml(comando['comandoSQL'],comando['datos'])
        verMensajeDatos('Operación UPDATE', respuesta_sql)
        stop()
        pass
    elif opcion == 4: #Borrar
        comando_sql = "SELECT rowid,* FROM productos;"
        respuesta_sql = objDB.ejecutar_dml(comando_sql)
        verTablaDatos('Resgistros Activos:', etiquetas_producto,
respuesta_sql)
        idReg = captarInt('Cual número de registro desea eliminar?
')

        comando_sql = "DELETE FROM productos WHERE rowid=?;"
        respuesta_sql = objDB.ejecutar_dml(comando_sql, [idReg])
        verMensajeDatos('Operación DELETE', respuesta_sql)
        stop()
        pass

objDB.cerrarConexion()
verMensajeDatos('Cerrar Aplicación', 'Gracias.')
```



```
if __name__ == '__main__':
    main()
```




El programa tiene un ciclo infinito hasta que el usuario digite 0.

Se crea una instancia de la clase SQL para el manejo de SQLite

```
nombreBaseDatos = 'inventario'  
objDB = SQL(nombreBaseDatos)
```

El ciclo infinito **while True:** carga el menú principal y espera cual opción es para ejecutar las instrucciones correspondientes.

Ya lo tienes, puedes hacer las modificaciones que desees para tu necesidad en particular.