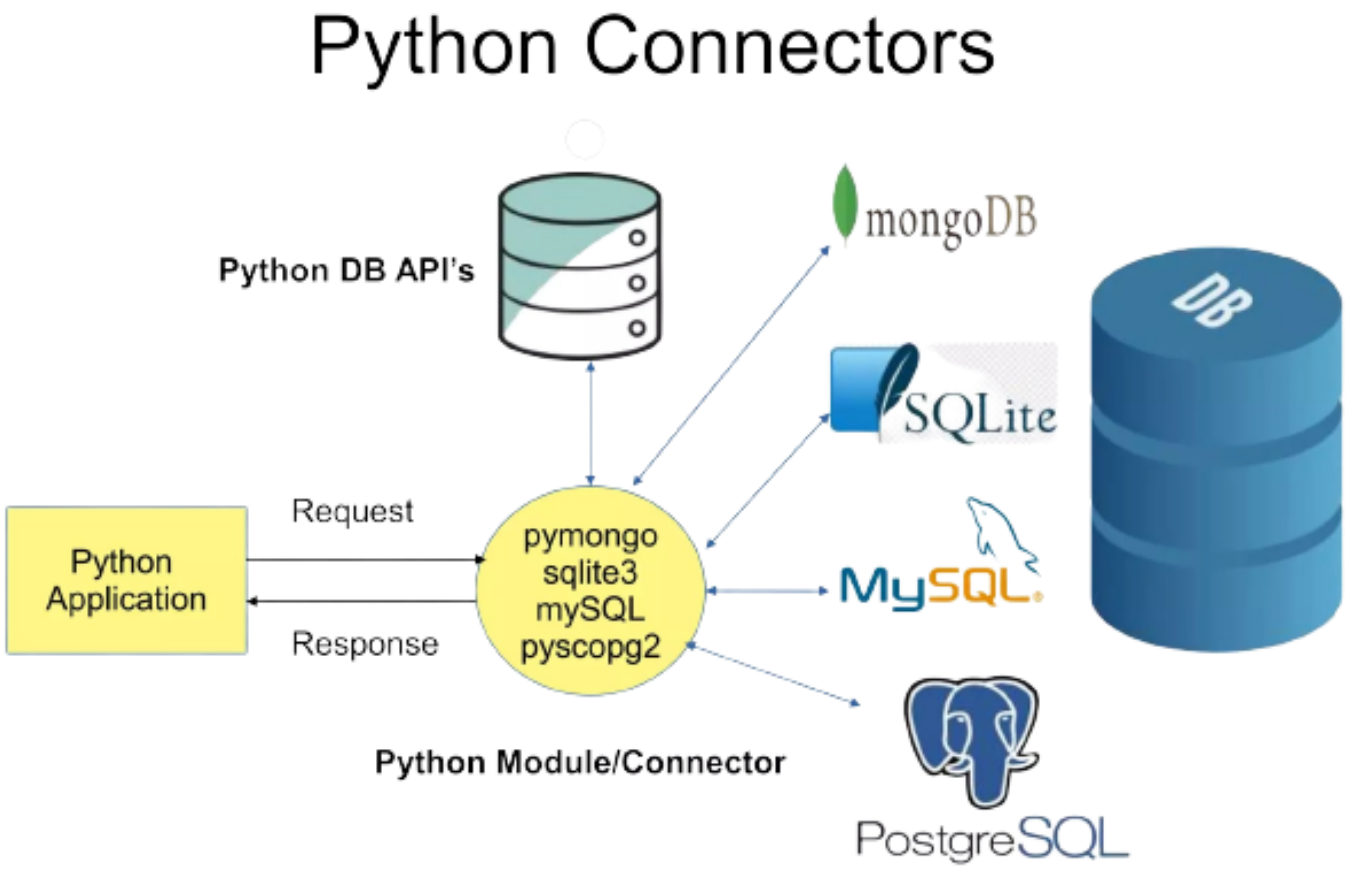
TECNOLOGÍAS DE INFORMACIÓN Y CIBERSEGURIDAD

PROGRAMACIÓN ORIENTADA A OBJETO (POO)

Integración con Bases de Datos

APIs DE CONEXIÓN A BASE DE DATOS

- Python ofrece múltiples APIs de conexión a diferentes bases de datos. A continuación, se presentan ejemplos utilizando algunas de las principales APIs para conectarse a bases de datos SQL y NoSQL.



¿QUÉ ES UNA API?

- Una API (Interfaz de Programación de Aplicaciones) es un conjunto de reglas y protocolos que permiten que distintas aplicaciones y sistemas se comuniquen entre sí de manera estandarizada.
- Proporciona una forma de acceder y utilizar funcionalidades de un software o servicio de manera controlada y segura, sin necesidad de conocer los detalles internos de su implementación.
- En pocas palabras, una API es como un intermediario que facilita la interacción y el intercambio de información entre diferentes componentes de software.

API DE CONEXIÓN A BASES DE DATOS SQL - SQLite (SQLite3)

```python
#SQLite es una base de datos SQL incorporada en Python.
#No se requiere instalación adicional.
import sqlite3

# Conexión a la base de datos o creación si no existe
conn = sqlite3.connect("mi_base_de_datos.db")

# Crear un cursor para ejecutar consultas SQL
cursor = conn.cursor()

# Ejemplo de creación de una tabla
cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY,
nombre TEXT, edad INTEGER)")

# Insertar datos en la tabla
cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES (?, ?)", ("Juan", 30))

# Realizar consulta
cursor.execute("SELECT * FROM usuarios")
resultados = cursor.fetchall()
print(resultados)

# Cerrar la conexión
conn.commit()
conn.close()
```

API DE CONEXIÓN A BASES DE DATOS SQL – MySQL (MySQL Connector)

```
#Para conectarse a una base de datos MySQL, necesitarás instalar el paquete mysql-connector-python. Puedes instalarlo utilizando pip install mysql-connector-python.
import mysql.connector

# Conexión a la base de datos
conn = mysql.connector.connect(
    host="localhost",
    user="usuario",
    password="contraseña",
    database="mi_base_de_datos"
)

# Crear un cursor para ejecutar consultas SQL
cursor = conn.cursor()
# Ejemplo de creación de una tabla
cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (id INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(255), edad INT)")

# Insertar datos en la tabla
cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES (%s, %s)", ("Pedro", 25))

# Realizar consulta
cursor.execute("SELECT * FROM usuarios")
resultados = cursor.fetchall()
print(resultados)

# Cerrar la conexión
conn.commit()
conn.close()
```

API DE CONEXIÓN A BASES DE DATOS SQL - PostgreSQL (psycopg2)

```
#PostgreSQL es un sistema de gestión de bases de datos SQL muy potente y de código abierto que se puede utilizar con Python mediante el paquete psycopg2.
import psycopg2

# Conexión a la base de datos o creación si no existe
conn = psycopg2.connect(
    host="localhost",
    user="tu_usuario",
    password="tu_contraseña",
    database="tu_base_de_datos"
)
# Crear un cursor para ejecutar consultas SQL
cursor = conn.cursor()

# Ejemplo de creación de una tabla
cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (id SERIAL PRIMARY KEY, nombre TEXT, edad INTEGER)")

# Insertar datos en la tabla
cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES (%s, %s)", ("Juan", 30))

# Realizar consulta
cursor.execute("SELECT * FROM usuarios")
resultados = cursor.fetchall()
print(resultados)

# Cerrar la conexión
conn.commit()
cursor.close()
conn.close()
```

API DE CONEXIÓN A BASES DE DATOS NoSQL - MongoDB (PyMongo)

```
#Para conectarse a MongoDB, necesitarás instalar el paquete pymongo. Puedes instalarlo utilizando pip install pymongo.
import pymongo

# Conexión a la base de datos
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Acceder a una base de datos
db = client["mi_base_de_datos"]

# Acceder a una colección (tabla)
usuarios = db["usuarios"]

# Ejemplo de inserción de un documento
usuario = {"nombre": "Ana", "edad": 28}
usuarios.insert_one(usuario)

# Realizar consulta
resultado = usuarios.find_one({"nombre": "Ana"})
print(resultado)

# Cerrar la conexión
client.close()
```

CRUD BÁSICO SQLite (SQLite3): CONEXIÓN Y CREAR

```
#Importamos el módulo sqlite3
import sqlite3

#Creamos la función para conectarnos a la base de datos sqlite3
def conectar():
    conn = sqlite3.connect("mi_base_de_datos.db")
    return conn

#Creamos la función para crear una tabla usuarios
def crear_tabla(conn):
    cursor = conn.cursor()
    cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (id INTEGER PRIMARY KEY,
nombre TEXT, edad INTEGER)")
    conn.commit()
    cursor.close()

#Creamos la función para crear un usuario en la tabla usuarios
def crear_usuario(conn, nombre, edad):
    cursor = conn.cursor()
    cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES (?, ?)", (nombre,
edad))
    conn.commit()
    cursor.close()
```

CRUD BÁSICO SQLite (SQLite3): LEER, ACTUALIZAR Y ELIMINAR

```
#Creamos la función para leer los usuarios de la tabla usuarios
def leer_usuarios(conn):
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM usuarios")
    usuarios = cursor.fetchall()
    cursor.close()
    return usuarios

#Creamos la función para actualizar un usuario de la tabla usuarios
def actualizar_usuario(conn, id_usuario, nombre, edad):
    cursor = conn.cursor()
    cursor.execute("UPDATE usuarios SET nombre=?, edad=? WHERE id=?",
(nombre, edad, id_usuario))
    conn.commit()
    cursor.close()

#Creamos la función para eliminar un usuario de la tabla usuarios
def eliminar_usuario(conn, id_usuario):
    cursor = conn.cursor()
    cursor.execute("DELETE FROM usuarios WHERE id=?", (id_usuario,))
    conn.commit()
    cursor.close()
```

CRUD BÁSICO SQLite (SQLite3): MENÚ

```
#Creamos la función para mostrar un menú
def mostrar_menu():
    print("\n--- Menú ---")
    print("1. Crear usuario")
    print("2. Leer usuarios")
    print("3. Actualizar usuario")
    print("4. Eliminar usuario")
    print("5. Salir")
```

CRUD BÁSICO SQLite (SQLite3): MAIN

```python
#Creamos la función principal para ejecutar el código
def main():
    conn = conectar()
    crear_tabla(conn)
    while True:
        mostrar_menu()
        opcion = input("Ingrese el número de la opción deseada: ")
        if opcion == "1":
            nombre = input("Ingrese el nombre del usuario: ")
            edad = int(input("Ingrese la edad del usuario: "))
            crear_usuario(conn, nombre, edad)
            print("Usuario creado exitosamente.")
        elif opcion == "2":
            usuarios = leer_usuarios(conn)
            if usuarios:
                print("\n--- Usuarios ---")
                for usuario in usuarios:
                    print(f"ID: {usuario[0]}, Nombre: {usuario[1]}, Edad: {usuario[2]}")
            else:
                print("No hay usuarios registrados.")
        elif opcion == "3":
            id_usuario = int(input("Ingrese el ID del usuario a actualizar: "))
            nombre = input("Ingrese el nuevo nombre del usuario: ")
            edad = int(input("Ingrese la nueva edad del usuario: "))
            actualizar_usuario(conn, id_usuario, nombre, edad)
            print("Usuario actualizado exitosamente.")
        elif opcion == "4":
            id_usuario = int(input("Ingrese el ID del usuario a eliminar: "))
            eliminar_usuario(conn, id_usuario)
            print("Usuario eliminado exitosamente.")
        elif opcion == "5":
            print("¡Hasta luego!")
            break
        else:
            print("Opción inválida. Por favor, ingrese una opción válida.")
    conn.close()
```

CRUD BÁSICO SQLite (SQLite3): VALIDACIÓN EJECUCIÓN

```
#Añadimos una validación para que se ejecute automáticamente
sólo si es el archivo principal
if __name__ == "__main__":
    main()
```

CRUD BÁSICO MySQL (MySQL Connector): CONEXIÓN Y CREAR

```python
#Importamos el módulo mysql.connector
import mysql.connector

#Creamos la función para conectarnos a la base de datos MySQL
def conectar():
    conn = mysql.connector.connect(
        host="localhost",
        user="tu_usuario",
        password="tu_contraseña",
        database="tu_base_de_datos"
    )
    return conn

#Creamos la función para crear un usuario en la tabla usuarios
def crear_usuario(conn, nombre, edad):
    cursor = conn.cursor()
    consulta = "INSERT INTO usuarios (nombre, edad) VALUES (%s, %s)"
    datos = (nombre, edad)
    cursor.execute(consulta, datos)
    conn.commit()
    cursor.close()
```

CRUD BÁSICO MySQL (MySQL Connector): LEER, ACTUALIZAR Y ELIMINAR

```python
#Creamos la función para leer usuarios desde la tabla usuarios
def leer_usuarios(conn):
    cursor = conn.cursor()
    consulta = "SELECT * FROM usuarios"
    cursor.execute(consulta)
    usuarios = cursor.fetchall()
    cursor.close()
    return usuarios

#Creamos la función para actualizar un usuario de la tabla usuarios
def actualizar_usuario(conn, id_usuario, nombre, edad):
    cursor = conn.cursor()
    consulta = "UPDATE usuarios SET nombre=%s, edad=%s WHERE id=%s"
    datos = (nombre, edad, id_usuario)
    cursor.execute(consulta, datos)
    conn.commit()
    cursor.close()

#Creamos la función para eliminar un usuario de la tabla usuarios
def eliminar_usuario(conn, id_usuario):
    cursor = conn.cursor()
    consulta = "DELETE FROM usuarios WHERE id=%s"
    datos = (id_usuario,)
    cursor.execute(consulta, datos)
    conn.commit()
    cursor.close()
```

CRUD BÁSICO MySQL (MySQL Connector): MENÚ

```
#Creamos la función para mostrar un menú
def mostrar_menu():
    print("\n--- Menú ---")
    print("1. Crear usuario")
    print("2. Leer usuarios")
    print("3. Actualizar usuario")
    print("4. Eliminar usuario")
    print("5. Salir")
```

CRUD BÁSICO MySQL (MySQL Connector): MAIN

```python
#Creamos la función principal para ejecutar el código
def main():
    conn = conectar()
    while True:
        mostrar_menu()
        opcion = input("Ingrese el número de la opción deseada: ")
        if opcion == "1":
            nombre = input("Ingrese el nombre del usuario: ")
            edad = int(input("Ingrese la edad del usuario: "))
            crear_usuario(conn, nombre, edad)
            print("Usuario creado exitosamente.")
        elif opcion == "2":
            usuarios = leer_usuarios(conn)
            if usuarios:
                print("\n--- Usuarios ---")
                for usuario in usuarios:
                    print(f"ID: {usuario[0]}, Nombre: {usuario[1]}, Edad: {usuario[2]}")
            else:
                print("No hay usuarios registrados.")
        elif opcion == "3":
            id_usuario = int(input("Ingrese el ID del usuario a actualizar: "))
            nombre = input("Ingrese el nuevo nombre del usuario: ")
            edad = int(input("Ingrese la nueva edad del usuario: "))
            actualizar_usuario(conn, id_usuario, nombre, edad)
            print("Usuario actualizado exitosamente.")
        elif opcion == "4":
            id_usuario = int(input("Ingrese el ID del usuario a eliminar: "))
            eliminar_usuario(conn, id_usuario)
            print("Usuario eliminado exitosamente.")
        elif opcion == "5":
            print("¡Hasta luego!")
            break
        else:
            print("Opción inválida. Por favor, ingrese una opción válida.")
    conn.close()
```

CRUD BÁSICO MySQL (MySQL Connector): VALIDACIÓN EJECUCIÓN

```
#Añadimos una validación para que se ejecute automáticamente
sólo si es el archivo principal
if __name__ == "__main__":
    main()
```

CRUD BÁSICO PostgreSQL (psycopg2): CONEXIÓN Y CREAR

```python
#Importamos el módulo psycopg2
import psycopg2

#Creamos la función para conectarnos a la base de datos PostgreSQL
def conectar():
    try:
        conn = psycopg2.connect(
            host="localhost",
            user="tu_usuario",
            password="tu_contraseña",
            database="tu_base_de_datos"
        )
        return conn
    except psycopg2.Error as e:
        print("Error al conectar a la base de datos:", e)
        return None

#Creamos la función para crear la tabla usuarios
def crear_tabla(conn):
    cursor = conn.cursor()
    cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (id SERIAL PRIMARY KEY, nombre TEXT, edad
INTEGER)")
    conn.commit()
    cursor.close()

#Creamos la función para crear un usuario en la tabla usuarios
def crear_usuario(conn, nombre, edad):
    cursor = conn.cursor()
    cursor.execute("INSERT INTO usuarios (nombre, edad) VALUES (%s, %s)", (nombre, edad))
    conn.commit()
    cursor.close()
```

CRUD BÁSICO PostgreSQL (psycopg2): LEER, ACTUALIZAR Y ELIMINAR

```python
#Creamos la función para leer usuarios desde la tabla usuarios
def leer_usuarios(conn):
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM usuarios")
    usuarios = cursor.fetchall()
    cursor.close()
    return usuarios

#Creamos la función para actualizar un usuario de la tabla usuarios
def actualizar_usuario(conn, id_usuario, nombre, edad):
    cursor = conn.cursor()
    cursor.execute("UPDATE usuarios SET nombre=%s, edad=%s WHERE
id=%s", (nombre, edad, id_usuario))
    conn.commit()
    cursor.close()

#Creamos la función para eliminar un usuario de la tabla usuarios
def eliminar_usuario(conn, id_usuario):
    cursor = conn.cursor()
    cursor.execute("DELETE FROM usuarios WHERE id=%s", (id_usuario,))
    conn.commit()
    cursor.close()
```

CRUD BÁSICO PostgreSQL (psycopg2): MENÚ

```
#Creamos la función para mostrar un menú
def mostrar_menu():
    print("\n--- Menú ---")
    print("1. Crear usuario")
    print("2. Leer usuarios")
    print("3. Actualizar usuario")
    print("4. Eliminar usuario")
    print("5. Salir")
```

CRUD BÁSICO PostgreSQL (psycopg2): MAIN

```python
#Creamos la función principal para ejecutar el código
def main():
    conn = conectar()
    if conn is None:
        return
    crear_tabla(conn)
    while True:
        mostrar_menu()
        opcion = input("Ingrese el número de la opción deseada: ")
        if opcion == "1":
            nombre = input("Ingrese el nombre del usuario: ")
            edad = int(input("Ingrese la edad del usuario: "))
            crear_usuario(conn, nombre, edad)
            print("Usuario creado exitosamente.")
        elif opcion == "2":
            usuarios = leer_usuarios(conn)
            if usuarios:
                print("\n--- Usuarios ---")
                for usuario in usuarios:
                    print(f"ID: {usuario[0]}, Nombre: {usuario[1]}, Edad: {usuario[2]}")
            else:
                print("No hay usuarios registrados.")
        elif opcion == "3":
            id_usuario = int(input("Ingrese el ID del usuario a actualizar: "))
            nombre = input("Ingrese el nuevo nombre del usuario: ")
            edad = int(input("Ingrese la nueva edad del usuario: "))
            actualizar_usuario(conn, id_usuario, nombre, edad)
            print("Usuario actualizado exitosamente.")
        elif opcion == "4":
            id_usuario = int(input("Ingrese el ID del usuario a eliminar: "))
            eliminar_usuario(conn, id_usuario)
            print("Usuario eliminado exitosamente.")
        elif opcion == "5":
            print("¡Hasta luego!")
            break
        else:
            print("Opción inválida. Por favor, ingrese una opción válida.")
    conn.close()
```

CRUD BÁSICO PostgreSQL (psycopg2): VALIDACIÓN EJECUCIÓN

```
#Añadimos una validación para que se ejecute automáticamente
sólo si es el archivo principal
if __name__ == "__main__":
    main()
```

CRUD BÁSICO MongoDB (PyMongo): CONEXIÓN Y CREAR

```python
#Importamos el módulo pymongo
import pymongo

#Creamos la función para conectarnos a la base de datos MySQL
def conectar():
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    db = client["mi_base_de_datos"]
    return db

#Creamos la función para crear un usuario en la tabla usuarios
def crear_usuario(db, nombre, edad):
    usuarios = db["usuarios"]
    usuario = {"nombre": nombre, "edad": edad}
    usuarios.insert_one(usuario)
```

CRUD BÁSICO MongoDB (PyMongo): LEER, ACTUALIZAR Y ELIMINAR

```python
#Creamos la función para leer usuarios desde la tabla usuarios
def leer_usuarios(db):
    usuarios = db["usuarios"]
    return list(usuarios.find())

#Creamos la función para actualizar un usuario de la tabla
usuarios
def actualizar_usuario(db, id_usuario, nombre, edad):
    usuarios = db["usuarios"]
    usuarios.update_one({"_id": id_usuario}, {"$set":
{"nombre": nombre, "edad": edad}})

#Creamos la función para eliminar un usuario de la tabla
usuarios
def eliminar_usuario(db, id_usuario):
    usuarios = db["usuarios"]
    usuarios.delete_one({"_id": id_usuario})
```

CRUD BÁSICO MongoDB (PyMongo): MENÚ

```
#Creamos la función para mostrar un menú
def mostrar_menu():
    print("\n--- Menú ---")
    print("1. Crear usuario")
    print("2. Leer usuarios")
    print("3. Actualizar usuario")
    print("4. Eliminar usuario")
    print("5. Salir")
```

CRUD BÁSICO MongoDB (PyMongo): MAIN

```python
#Creamos la función principal para ejecutar el código
def main():
    db = conectar()
    while True:
        mostrar_menu()
        opcion = input("Ingrese el número de la opción deseada: ")
        if opcion == "1":
            nombre = input("Ingrese el nombre del usuario: ")
            edad = int(input("Ingrese la edad del usuario: "))
            crear_usuario(db, nombre, edad)
            print("Usuario creado exitosamente.")
        elif opcion == "2":
            usuarios = leer_usuarios(db)
            if usuarios:
                print("\n--- Usuarios ---")
                for usuario in usuarios:
                    print(f"ID: {usuario['_id']}, Nombre: {usuario['nombre']}, Edad: {usuario['edad']}")
            else:
                print("No hay usuarios registrados.")
        elif opcion == "3":
            id_usuario = input("Ingrese el ID del usuario a actualizar: ")
            nombre = input("Ingrese el nuevo nombre del usuario: ")
            edad = int(input("Ingrese la nueva edad del usuario: "))
            actualizar_usuario(db, id_usuario, nombre, edad)
            print("Usuario actualizado exitosamente.")
        elif opcion == "4":
            id_usuario = input("Ingrese el ID del usuario a eliminar: ")
            eliminar_usuario(db, id_usuario)
            print("Usuario eliminado exitosamente.")
        elif opcion == "5":
            print("¡Hasta luego!")
            break
        else:
            print("Opción inválida. Por favor, ingrese una opción válida.")
```

CRUD BÁSICO MongoDB (PyMongo): VALIDACIÓN EJECUCIÓN

```
#Añadimos una validación para que se ejecute automáticamente
sólo si es el archivo principal
if __name__ == "__main__":
    main()
```

ENCRIPTADO DE DATOS: HASHLIB Y CRYPTOGRAPHY

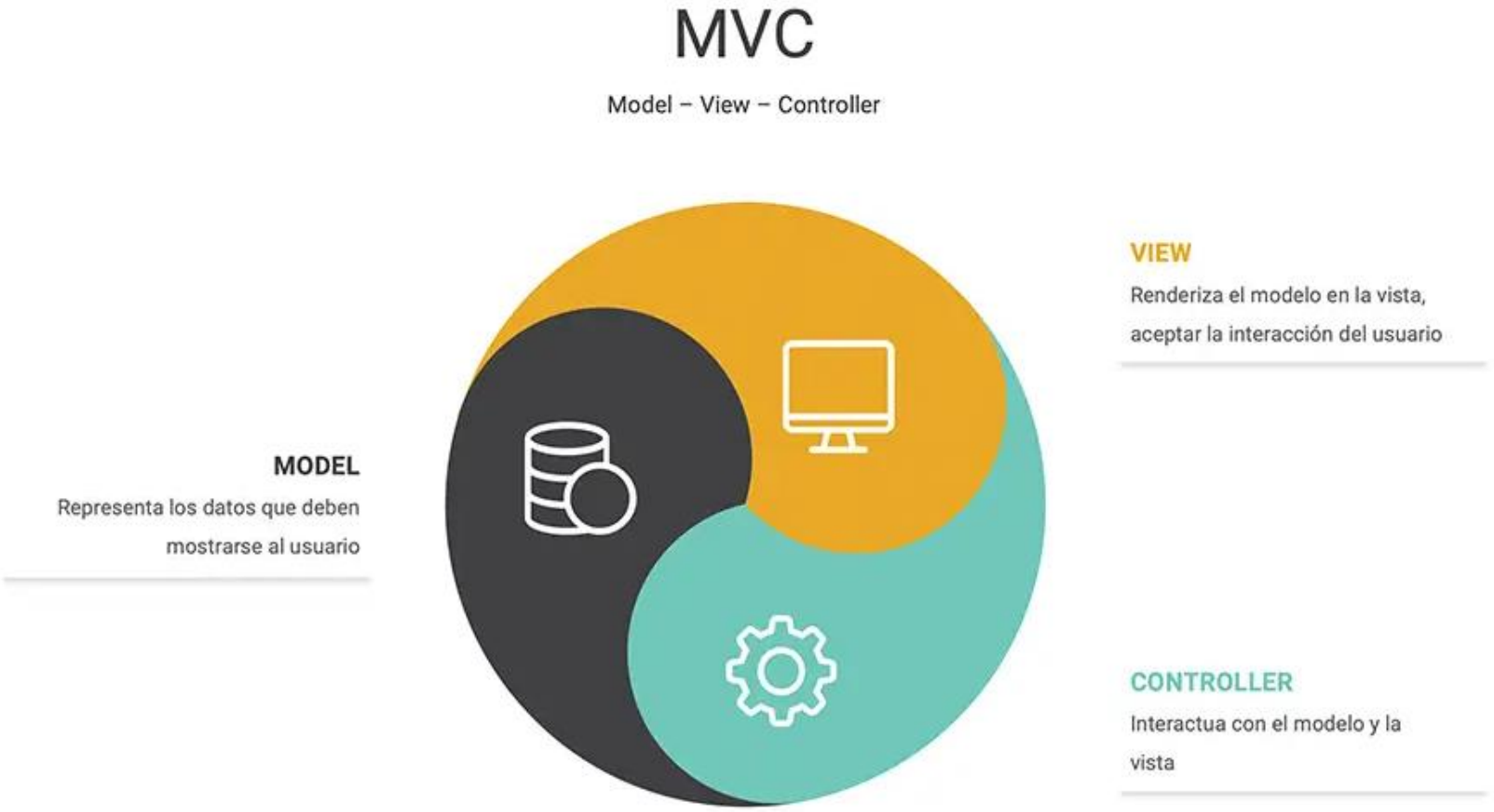
```
#Aquí tenemos un código básico en Python que muestra distintos tipos de encriptado de datos utilizando las bibliotecas hashlib y cryptography
import hashlib
from cryptography.fernet import Fernet

# Encriptado utilizando hashlib (hash)
def encriptar_con_hash(texto):
    hashed_text = hashlib.sha256(texto.encode()).hexdigest()
    return hashed_text

# Encriptado utilizando Fernet (simétrico)
def generar_clave():
    return Fernet.generate_key()

def encriptar_con_fernet(texto, clave):
    f = Fernet(clave)
    encrypted_text = f.encrypt(texto.encode())
    return encrypted_text

def desencriptar_con_fernet(texto_encriptado, clave):
    f = Fernet(clave)
    decrypted_text = f.decrypt(texto_encriptado).decode()
    return decrypted_text
```

ENCRIPTADO DE DATOS: UTILIZANDO LAS FUNCIONES CREADAS

```
# Ejemplo de uso
if __name__ == "__main__":
    texto_original = "Hola, este es un texto secreto."

    # Encriptado con hashlib (hash)
    hashed_texto = encriptar_con_hash(texto_original)
    print("Encriptado con hashlib:", hashed_texto)

    # Encriptado con Fernet (simétrico)
    clave_secreta = generar_clave()
    texto_encriptado = encriptar_con_fernet(texto_original, clave_secreta)
    print("Texto encriptado con Fernet:", texto_encriptado)

    texto_desencriptado = desencriptar_con_fernet(texto_encriptado, clave_secret
a)
    print("Texto desencriptado:", texto_desencriptado)
```

CRUD USANDO CLASES Y OBJETOS: PATRÓN MVC

- MVC es un patrón de diseño arquitectónico que separa una aplicación en tres componentes principales: Modelo, Vista y Controlador.
- Estos componentes trabajan juntos para mantener una estructura organizada y facilitar el desarrollo y mantenimiento de la aplicación.



CRUD USANDO CLASES Y OBJETOS: MAIN

```
#Importamos los módulos correspondientes al MVC
from model import Usuario
from view import Vista
from controller import Controlador

# Crear un objeto de cada componente (Modelo, Vista y Controlador)
usuario = Usuario()
vista = Vista()
controlador = Controlador(usuario, vista)

# Iniciar la aplicación
controlador.iniciar()
```

CRUD USANDO CLASES Y OBJETOS: MODEL

```python
#En este usaremos una base de datos MySQL, así que Importamos el módulo correspondiente
import mysql.connector
class Usuario:
    def __init__(self):
        self.conn = mysql.connector.connect(
            host="localhost",
            user="tu_usuario",
            password="tu_contraseña",
            database="tu_base_de_datos"
        )
        self.cursor = self.conn.cursor()
    def crear_usuario(self, nombre, edad):
        consulta = "INSERT INTO usuarios (nombre, edad) VALUES (%s, %s)"
        datos = (nombre, edad)
        self.cursor.execute(consulta, datos)
        self.conn.commit()
    def leer_usuarios(self):
        self.cursor.execute("SELECT * FROM usuarios")
        return self.cursor.fetchall()
    def actualizar_usuario(self, id_usuario, nombre, edad):
        consulta = "UPDATE usuarios SET nombre=%s, edad=%s WHERE id=%s"
        datos = (nombre, edad, id_usuario)
        self.cursor.execute(consulta, datos)
        self.conn.commit()
    def eliminar_usuario(self, id_usuario):
        consulta = "DELETE FROM usuarios WHERE id=%s"
        datos = (id_usuario,)
        self.cursor.execute(consulta, datos)
        self.conn.commit()
    def cerrar_conexion(self):
        self.cursor.close()
        self.conn.close()
```

CRUD USANDO CLASES Y OBJETOS: VIEW

```python
#Creamos la clase que usaremos para la interfaz
class Vista:
    def mostrar_menu(self):
        print("\n--- Menú ---")
        print("1. Crear usuario")
        print("2. Leer usuarios")
        print("3. Actualizar usuario")
        print("4. Eliminar usuario")
        print("5. Salir")


    def solicitar_datos_usuario(self):
        nombre = input("Ingrese el nombre del usuario: ")
        edad = int(input("Ingrese la edad del usuario: "))
        return nombre, edad


    def mostrar_usuarios(self, usuarios):
        if usuarios:
            print("\n--- Usuarios ---")
            for usuario in usuarios:
                print(f"ID: {usuario[0]}, Nombre: {usuario[1]}, Edad: {usuario[2]}")
        else:
            print("No hay usuarios registrados.")


    def solicitar_id_usuario(self):
        return int(input("Ingrese el ID del usuario: "))


    def mostrar_mensaje(self, mensaje):
        print(mensaje)
```

CRUD USANDO CLASES Y OBJETOS: CONTROLLER

```python
#Creamos la clase que usaremos para el controlador
class Controlador:
    def __init__(self, usuario, vista):
        self.usuario = usuario
        self.vista = vista
    def iniciar(self):
        while True:
            self.vista.mostrar_menu()
            opcion = input("Ingrese el número de la opción deseada: ")
            if opcion == "1":
                nombre, edad = self.vista.solicitar_datos_usuario()
                self.usuario.crear_usuario(nombre, edad)
                self.vista.mostrar_mensaje("Usuario creado exitosamente.")
            elif opcion == "2":
                usuarios = self.usuario.leer_usuarios()
                self.vista.mostrar_usuarios(usuarios)
            elif opcion == "3":
                id_usuario = self.vista.solicitar_id_usuario()
                nombre, edad = self.vista.solicitar_datos_usuario()
                self.usuario.actualizar_usuario(id_usuario, nombre, edad)
                self.vista.mostrar_mensaje("Usuario actualizado exitosamente.")
            elif opcion == "4":
                id_usuario = self.vista.solicitar_id_usuario()
                self.usuario.eliminar_usuario(id_usuario)
                self.vista.mostrar_mensaje("Usuario eliminado exitosamente.")
            elif opcion == "5":
                self.usuario.cerrar_conexion()
                print("¡Hasta luego!")
                break
            else:
                print("Opción inválida. Por favor, ingrese una opción válida.")
```