Herencias
Entonces como les venía diciendo lo él entregable de que tienen que hacer el programa funcionando en consola con conexión de base de datos de su proyecto yo voy a estar haciendo como una especie de introducción o tutorial de paso a paso de lo que deben de hacer todavía no le he terminado porque estaba ocupado haciendo otras cositas está terminando otro tutorial para que quede muy claro que lo que deben de hacer dentro del proyecto de ustedes cierto para que se vea muy funcional y hoy vamos a estar viendo la parte de herencias y lo que alcancemos en temas Entonces vamos a estar haciendo el trabajo conjunto entre el diagrama de clases y cómo lo vamos a estar llevando en la programación para que ustedes vayan desarrollando la parte lógica y ustedes tengan prácticamente de entregable de fase porque ustedes se lo van a estar solicitando el diagrama uml porque literal esto debió haber quedado el trimestre pasado pero no dio tiempo entonces que lo que estamos haciendo crearlas ya nos dimos cuenta que hay unos trucos que debemos de tenerles la matriz de requerimientos ya pulidita para que lo que con la base de datos empezamos a identificar puedan terminar de darle la funcionalidades correctas a los diagramas que ustedes están prácticamente diseñando y van a terminar en codificar porque di yango diyango toda esta parte de framework está basado en programación orientada objetos entonces debemos de tener muy bien concertado ese conocimiento

https://github.com/dileofrancoj/elite-web-developer-books/blob/main/el-libro-negro-del-programador.pdf

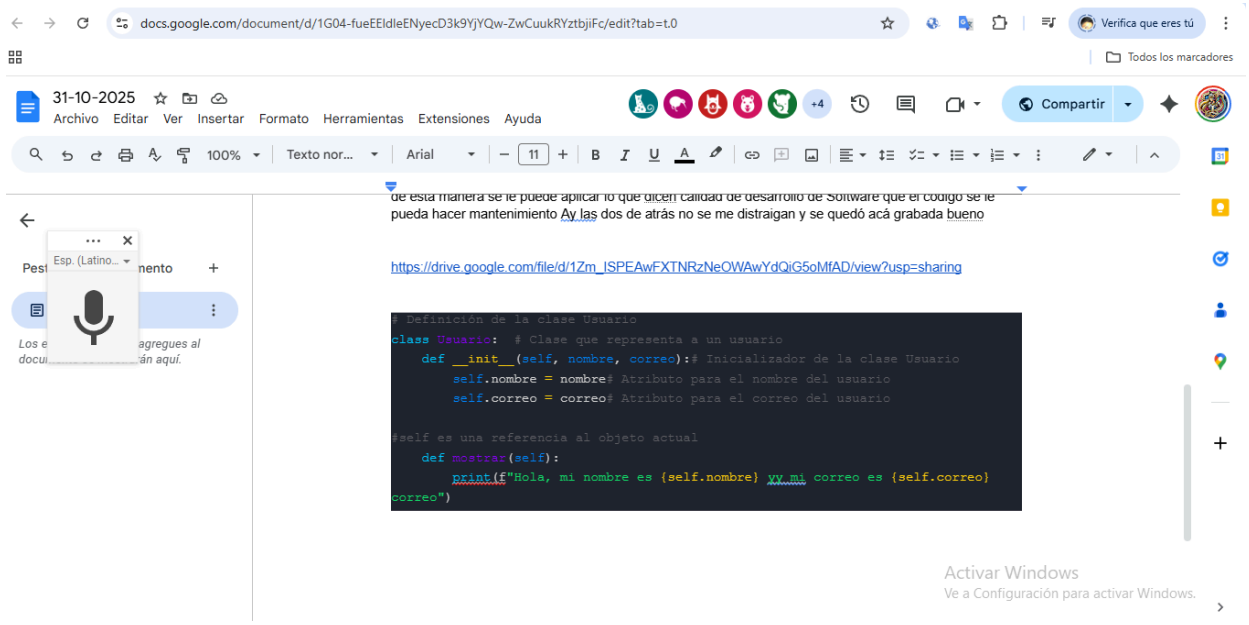
https://ellibrodepython.com/python-pep8
Entonces en conclusión documentar el código debe de estar implementando todo lo que está expuesto con código limpio ahí les indiqué en python cómo se llama es llamar los las variables las clases las funciones mecánicamente entendibles de que la persona que esté leyendo el código lo pueda entender de esta manera se le puede aplicar lo que dicen calidad de desarrollo de Software que el código se le pueda hacer mantenimiento Ay las dos de atrás no se me distraigan y se quedó acá grabada bueno

https://drive.google.com/file/d/1Zm_ISPEAwFXTNRzNeOWAwYdQiG5oMfAD/view?usp=sharing

```python
# Definición de la clase Usuario
class Usuario:  # Clase que representa a un usuario
    def __init__(self, nombre, correo):# Inicializador de la clase Usuario
        self.nombre = nombre# Atributo para el nombre del usuario
        self.correo = correo# Atributo para el correo del usuario

#self es una referencia al objeto actual
```

```
    def mostrar(self):
        print(f"Hola, mi nombre es {self.nombre} yy mi correo es
{self.correo} correo")
```




```
# clase hereda de Usuario a vendedor
class Vendedor(Usuario):# Clase que representa a un vendedor, hereda de
Usuario
    def __init__(self, nombre, correo, tienda):# Inicializador de la clase
Vendedor
        super().__init__(nombre, correo)# Llama al inicializador de la
clase base Usuario
        self.tienda = tienda# Atributo para la tienda del vendedor

    def mostrar(self):# Método para mostrar la información del vendedor
        super().mostrar()# Llama al método mostrar de la clase base
Usuario
        print(f"Vendo en la tienda: {self.tienda}")# Imprime la tienda del
vendedor
```

```
# ejemplo de objeto usuario, se escribe en minuscula
usuario1 = Usuario("Ana", "ana@example.com")# Crea una instancia de
Usuario
usuario1.mostrar()# Llama al método mostrar del usuario1
```

```
# ejemplo de objeto vendedor, se escribe en minuscula
vendedor1 = Vendedor("Luis", "luis@example.com", "Tienda de Ropa")# Crea
una instancia de Vendedor
vendedor1.mostrar()# Llama al método mostrar del vendedor1
```

```
class Administrador(Usuario):# Clase que representa a un administrador,
hereda de Usuario
    #este es el metodo que aplica el polimorfismo
    def mostrar(self):# Método para mostrar la información del
administrador
        return f"Administrador: {self.nombre}, Correo: {self.correo}"#
Retorna una cadena con el nombre y correo del administrador
```

correcciones del codigo

```
class Administrador(Usuario):# Clase que representa a un administrador,
hereda de Usuario
    #este es el metodo que aplica el polimorfismo
    def mostrar_info(self):# Método para mostrar la información del
administrador
        return f"Administrador: {self.nombre}, Correo: {self.correo}"#
Retorna una cadena con el nombre y correo del administrador

# ejemplo de uso de la clase Administrador y su metodo mostrar
def mostrar_administrador(usuario: Usuario):# Función que muestra la
información del administrador
    print(usuario.mostrar_info())  # Llama al método mostrar_info del
administrador
```

codigo completo:

```
# Definición de la clase Usuario
class Usuario:  # Clase que representa a un usuario
    def __init__(self, nombre, correo):# Inicializador de la clase Usuario
        self.nombre = nombre# Atributo para el nombre del usuario
        self.correo = correo# Atributo para el correo del usuario

#self es una referencia al objeto actual
```

```
    def mostrar(self):# Método para mostrar la información del usuario
        print(f"Hola, mi nombre es {self.nombre} yy mi correo es
{self.correo} correo")# Imprime el nombre y correo del usuario

# clase hereda de Usuario a vendedor
class Vendedor(Usuario):# Clase que representa a un vendedor, hereda de
Usuario
    def __init__(self, nombre, correo, tienda):# Inicializador de la clase
Vendedor
        super().__init__(nombre, correo)# Llama al inicializador de la
clase base Usuario
        self.tienda = tienda# Atributo para la tienda del vendedor

    def mostrar(self):# Método para mostrar la información del vendedor
        super().mostrar()# Llama al método mostrar de la clase base
Usuario
        print(f"Vendo en la tienda: {self.tienda}")# Imprime la tienda del
vendedor
# ejemplo de objeto usuario, se escribe en minuscula
usuario1 = Usuario("Ana", "ana@example.com")# Crea una instancia de
Usuario
usuario1.mostrar()# Llama al método mostrar del usuario1

# ejemplo de objeto vendedor, se escribe en minuscula
vendedor1 = Vendedor("Luis", "luis@example.com", "Tienda de Ropa")# Crea
una instancia de Vendedor
vendedor1.mostrar()# Llama al método mostrar del vendedor1

# la Clase administrador hereda de Usuario y tiene un metodo mostrar que
aplica polimorfismo
class Administrador(Usuario):# Clase que representa a un administrador,
hereda de Usuario
    #este es el metodo que aplica el polimorfismo
    def mostrar_info(self):# Método para mostrar la información del
administrador
        return f"Administrador: {self.nombre}, Correo: {self.correo}"#
Retorna una cadena con el nombre y correo del administrador

# ejemplo de uso de la clase Administrador y su metodo mostrar
def mostrar_detalle(usuario: Usuario):# Función que muestra la información
```

```
    print(usuario.mostrar_info())  # Llama al método mostrar_info del
administrador

# instancias  vendedor y administrador objetos
vendedor2 = Vendedor("Carlos", "carlos@example.com", "Tienda de
Electrónica")# Crea una instancia de Vendedor
administrador1 = Administrador("Marta", "marta@example.com")
#Mostrar detalles del vendedor y administrador

mostrar_detalle(administrador1)  # Muestra la información del
administrador# Crea una instancia de Administrador
```