EXPLICACIÓN DEL CÓDIGO

MAIN.PY

1. `verificar_empleado()`

- Propósito: Verifica si el empleado ingresado (nombre e ID) existe en la base de datos de empleados.
- Detalles: Si los datos son correctos, verifica si el empleado es un jefe o no. Si es jefe, abre una ventana con opciones para consultar horarios de otros empleados; si no es jefe, abre una ventana con el horario del propio empleado.

2. `buscar_empleado_binario(nombre, id_empleado)`

- Propósito: Busca un empleado por su nombre e ID utilizando búsqueda binaria.
- Detalles: Los empleados están ordenados previamente por nombre con `merge_sort()`. La función busca el empleado en la lista ordenada y retorna el empleado correspondiente si la búsqueda tiene éxito, de lo contrario, retorna `None`.

3. `merge_sort(empleados)`

- Propósito: Ordena la lista de empleados alfabéticamente por nombre.
- Detalles: Implementa el algoritmo Merge Sort. Divide la lista en mitades, ordena cada mitad recursivamente y luego combina las mitades ordenadas en una lista final ordenada.

4. `ventana_jefe(empleado)`

- Propósito: Crea una ventana para el jefe con opciones para consultar los horarios de los empleados.
- Detalles: Muestra un saludo al jefe y un campo de entrada para seleccionar a un empleado y consultar su turno de hoy o su horario semanal.

5. `consultar_turno_jefe(combo_empleado, jefe, entry_dia)`

- Propósito: Permite al jefe consultar el turno de un empleado para un día específico.
- Detalles: Utiliza el nombre del empleado seleccionado y el día ingresado para buscar el turno del empleado. Si el turno existe, muestra la hora; de lo contrario, muestra un error.

6. `consultar_semana_jefe(combo_empleado, jefe, entry_dia)`

- Propósito: Muestra el horario completo semanal de un empleado.
- Detalles: Para el empleado seleccionado, muestra los turnos asignados para cada día de la semana. Si un día no tiene turno asignado, lo indica.

7. `ventana_empleado(empleado)`

- Propósito: Crea una ventana para el empleado donde puede consultar su propio horario.
- Detalles: Muestra un saludo al empleado y le permite consultar su turno de hoy o su horario semanal.

8. `consultar_turno_empleado(empleado, entry_dia)`

- Propósito: Permite al empleado consultar su turno de hoy.
- Detalles: Usa el día ingresado para buscar el turno del empleado en ese día específico. Si el turno existe, muestra la hora; de lo contrario, muestra un error.

9. `consultar_semana_empleado(empleado, entry_dia)`

- Propósito: Muestra el horario completo semanal de un empleado.
- Detalles: Similar a la función `consultar_semana_jefe`, pero en este caso el horario es consultado por el propio empleado.

10. Ventana Principal (`ventana_principal`)

- Propósito: Ventana de inicio donde el usuario ingresa su nombre y ID.
- Detalles: Contiene dos campos de entrada (nombre e ID) y un botón para verificar los datos ingresados. Si la verificación es exitosa, redirige al usuario a la ventana correspondiente (jefe o empleado).

EMPLEADO.PY

1. `class Empleado:`

- Propósito: Define una clase llamada `Empleado`, que tiene atributos y métodos asociados a un empleado.
- Detalles: Esta clase va a manejar la información básica de un empleado, como su nombre, ID y si es jefe o no.

2. `def __init__(self, nombre, id, es_jefe=False):`

- Propósito: Este es el método constructor de la clase `Empleado`. Se ejecuta cuando se crea una nueva instancia (objeto) de la clase `Empleado`.
- Detalles:
  - `nombre`: El nombre del empleado.
  - `id`: El ID único del empleado.
  - `es_jefe`: Un valor booleano que indica si el empleado es jefe o no. Por defecto, se establece como `False` (es decir, el empleado no es jefe a menos que se indique lo contrario).
- Qué hace: Asigna los valores de los parámetros a los atributos privados de la clase. Estos atributos se definen con el prefijo `_` para indicar que son internos a la clase.

3. `@property`

- Propósito: El decorador `@property` convierte un método en una propiedad de la clase. Esto significa que, en lugar de acceder a un valor directamente (como `empleado.nombre`), se llama al método `nombre()` sin necesidad de usar paréntesis.
- Detalles: Las propiedades permiten controlar cómo se obtiene o se establece el valor de un atributo, a la vez que proporcionan un acceso más controlado y más seguro.

4. `def nombre(self):`

- Propósito: Método que devuelve el valor del atributo `_nombre` del empleado.
- Qué hace: Este método simplemente devuelve el valor del nombre del empleado. Al ser una propiedad, puedes acceder al nombre usando `empleado.nombre` sin necesidad de paréntesis.

5. `def id(self):`

- Propósito: Método que devuelve el valor del atributo `_id` del empleado.

- Qué hace: Al igual que el método `nombre()`, este método devuelve el valor del ID del empleado. Puedes acceder a este valor usando `empleado.id`.

6. `def es_jefe(self):`

- Propósito: Método que devuelve el valor del atributo `_es_jefe`, que indica si el empleado es un jefe.
- Qué hace: Este método devuelve el valor booleano (`True` o `False`) que indica si el empleado es jefe o no. Se puede acceder usando `empleado.es_jefe`.

HORARIO.PY

1. `from datetime import time`

- Propósito: Importa la clase `time` del módulo `datetime`, que se usa para representar un punto en el tiempo (como las horas de inicio y fin de los turnos de trabajo).

2. `from empleado import Empleado`

- Propósito: Importa la clase `Empleado` desde el archivo `empleado.py`, que se utiliza para representar a los empleados de la farmacia.

3. `class Horario:`

- Propósito: Define la clase `Horario`, que maneja la información sobre los turnos de los empleados y sus horarios de trabajo.

4. `def __init__(self):`

- Propósito: Define el método constructor de la clase `Horario`. Este método se ejecuta cuando se crea una nueva instancia de la clase `Horario`.
- Qué hace: Inicializa los días de la semana y los turnos predefinidos mediante el método `_crear_turnos`.

5. `self.dias_semana = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]`

- Propósito: Crea una lista con los días de la semana.
- Qué hace: Esta lista se usará para referirse a los días de la semana a lo largo del código, como claves para los turnos de cada día.

6. `self.turnos_predefinidos = self._crear_turnos()`

- Propósito: Llama al método `_crear_turnos` para generar los turnos predefinidos para los empleados.
- Qué hace: Guarda el resultado de `_crear_turnos` en el atributo `turnos_predefinidos`, que es un diccionario con los turnos asignados a cada empleado por día.

7. `def _crear_turnos(self):`

- Propósito: Define un método privado para crear los turnos predefinidos de los empleados.
- Qué hace: Crea instancias de la clase `Empleado` y asigna un turno para cada uno de los días de la semana.

8. `empleado_1 = Empleado("Victoria", 101)` (y las siguientes instancias de empleados)

- Propósito: Crea instancias de la clase `Empleado` para representar a cada empleado de la farmacia.
- Qué hace: Crea empleados con nombres específicos e IDs únicos. El último empleado (`jefe`) tiene el atributo `es_jefe=True`, lo que lo identifica como jefe.

9. `self.empleados = [empleado_1, empleado_2, empleado_3, empleado_4, jefe]`

- Propósito: Guarda una lista de los empleados en el atributo `empleados`.
- Qué hace: Almacena todas las instancias de `Empleado` creadas en la lista `self.empleados`.

10. `return { ... }`

- Propósito: Retorna un diccionario que contiene los turnos de los empleados por día.
- Qué hace: El diccionario tiene los días de la semana como claves, y los valores son sub-diccionarios con empleados como claves y su turno asignado como valor (el turno es un par de objetos `time`, uno para la hora de inicio y otro para la hora de fin).

11. `def buscar_empleado(self, nombre, id):`

- Propósito: Define un método para buscar un empleado por nombre e ID.
- Qué hace: Recibe un nombre y un ID como parámetros, recorre la lista de empleados (`self.empleados`), y devuelve el empleado que coincida con ambos criterios (nombre e ID). Si no se encuentra, retorna `None`.