Explicación del Razonamiento y Justificación de la Solución

**1 - Gestión de Tienda**

En este documento, se describe el razonamiento detrás de la solución propuesta para la venta de un producto utilizando Python. Se detalla la decisión de utilizar Programación Orientada a Objetos (POO) y otras funcionalidades de Python para simplificar el proceso de venta y calcular el total y el cambio.

# Uso de Programación Orientada a Objetos (POO)

Se decidió utilizar POO para organizar el código de manera clara y modular. La clase 'Tienda' encapsula el comportamiento necesario para realizar una venta. Aunque en este caso solo se maneja un producto, el uso de una clase permite que el código sea fácilmente extensible en el futuro, por ejemplo, si se desea manejar un inventario completo o realizar múltiples ventas.

# Funcionalidades de Python Utilizadas

Se utilizaron las siguientes funcionalidades de Python en el código:

- \*\*Entrada de datos por teclado:\*\* Se utilizan las funciones `input` para permitir que el usuario ingrese el nombre del producto, el precio, la cantidad y el pago del cliente.

- \*\*Cálculos matemáticos:\*\* Se realizan operaciones simples de multiplicación y resta para calcular el total de la venta y el cambio que se debe entregar al cliente.

- \*\*Condicionales:\*\* Se utiliza un condicional `if-else` para verificar si el pago es suficiente para cubrir el total de la venta y, de ser así, calcular el vuelto; en caso contrario, se notifica al usuario.

# Justificación de las Decisiones

La decisión de utilizar POO en este caso fue tomada para mantener el código organizado y modular, lo que facilita su extensión en el futuro. Aunque el problema a resolver es sencillo, la estructura orientada a objetos proporciona una base sólida para manejar problemas más complejos. Además, el uso de funcionalidades básicas de Python como la entrada de datos por teclado y las operaciones matemáticas permite que el código sea fácil de entender y de modificar según sea necesario.

**2 - Gestión de Asistencia**

En este documento, se explica cómo se resolvió el problema de registrar la asistencia de un estudiante usando Python. Se detalla por qué se eligió Programación Orientada a Objetos (POO) y qué otras funciones de Python se usaron para facilitar el registro y la creación de reportes.

# Uso de Programación Orientada a Objetos (POO)

Se decidió usar POO para organizar el código de manera más clara. Se crearon dos clases: Estudiante y Docente. La clase Estudiante se encarga de guardar la información del estudiante, como el nombre y los registros de asistencia. La clase Docente gestiona el grupo de estudiantes, registra la asistencia y genera reportes.

# Funcionalidades de Python Utilizadas

Aquí están las principales funciones de Python que se usaron:

* **Entrada de datos por teclado:** Se utiliza la función input() para que el usuario pueda ingresar datos como la fecha, si el estudiante asistió o no, y la razón si hubo un permiso. Esto permite registrar la asistencia de manera interactiva.
* **Estructuras de datos:** Se emplean listas y diccionarios para organizar la información. Las listas mantienen un registro de las asistencias del estudiante, y los diccionarios ayudan a crear el reporte final asociando el nombre del estudiante con sus asistencias.
* **Condicionales:** Se usa if-else para manejar diferentes situaciones de asistencia. Esto permite clasificar si el estudiante asistió, estuvo ausente con permiso, o tuvo una inasistencia sin permiso.

# Justificación de las Decisiones

La decisión de usar POO se basó en la necesidad de mantener el código organizado y fácil de entender. La clase Estudiante maneja todo lo relacionado con el estudiante, y la clase Docente se encarga de las tareas relacionadas con el grupo de estudiantes. Esto facilita agregar nuevas funciones más adelante.

Las funciones básicas de Python, como input() y las estructuras de datos, hacen que el código sea fácil de seguir y de modificar. Aunque el problema es sencillo, la estructura orientada a objetos proporciona una base sólida para manejar problemas más complejos en el futuro.

**3 – Gestión de Reservas en un Hotel de Playa**

Se detalla el razonamiento detrás de la solución propuesta para gestionar reservas en un hotel de playa utilizando Python. Se explica la decisión de emplear Programación Orientada a Objetos (POO) y otras funcionalidades de Python, y se justifican las decisiones tomadas en el diseño del programa.

# Uso de Programación Orientada a Objetos (POO)

Se decidió utilizar POO para estructurar el código de manera clara y modular. Las principales clases definidas son Habitacion, Cliente, y Hotel. Cada clase se encarga de un aspecto específico del sistema:

* **Clase Habitacion:** Representa una habitación del hotel con su tipo y precio por noche. Esta clase permite manejar las propiedades de las habitaciones de forma independiente y reutilizable.
* **Clase Cliente:** Guarda la información del cliente, como el nombre, DUI, habitación seleccionada, número de noches y servicios extra solicitados. Esta clase facilita el manejo de los datos del cliente y el cálculo del total a pagar.
* **Clase Hotel:** Gestiona las habitaciones disponibles y los servicios extra que se pueden ofrecer. Permite agregar nuevas habitaciones y servicios, y mostrar las opciones disponibles al cliente.

# Funcionalidades de Python Utilizadas

El programa utiliza varias funcionalidades básicas de Python:

* **Entrada de datos por teclado:** Se emplea la función input() para que el usuario pueda ingresar información como el nombre del cliente, el DNI, la selección de la habitación, el número de noches, y los servicios extra. Esto permite una interacción dinámica con el usuario.
* **Estructuras de datos:** Se utilizan listas para almacenar las habitaciones y los servicios extra disponibles. También se utilizan diccionarios para gestionar los servicios extra con sus nombres y precios, lo que facilita el acceso a estos datos y su presentación al usuario.
* **Cálculos matemáticos:** Se realizan operaciones básicas como multiplicación y suma para calcular el total de la estancia y los servicios adicionales. Estos cálculos permiten determinar el costo total de la reserva y el monto que el cliente debe pagar.
* **Condicionales:** Se usan condicionales if-else para manejar la lógica de selección de servicios extra y la generación de la factura. Esto asegura que el cliente pueda agregar servicios adicionales y que se muestre un resumen correcto de los gastos.

# Justificación de las Decisiones

La elección de usar POO se basa en la necesidad de mantener el código organizado y modular. La estructura de clases permite separar las responsabilidades del programa, facilitando la comprensión y el mantenimiento del código. La clase Habitacion maneja los detalles de cada habitación, la clase Cliente gestiona la información del cliente y la reserva, y la clase Hotel administra las habitaciones y servicios disponibles.

**4 - Gestión de Pago**

El razonamiento detrás de la solución propuesta para gestionar la planilla de pago en una empresa que cuenta con dos tipos de empleados: aquellos con plaza fija y aquellos que trabajan por horas. Se detalla la decisión de utilizar Programación Orientada a Objetos (POO) y otras funcionalidades de Python para organizar y calcular los pagos correspondientes.

# Uso de Programación Orientada a Objetos (POO)

Se decidió utilizar POO para estructurar el código de manera que refleje claramente las diferencias entre los tipos de empleados. Al crear clases para los empleados, el código se mantiene modular y fácil de entender. Cada tipo de empleado tiene sus propias características y métodos para calcular su pago, lo que permite una fácil expansión o modificación del sistema en el futuro.

* **Clase Empleado:** Sirve como clase base para encapsular atributos comunes como el nombre y los años trabajados. También incluye el método para calcular el bono, aplicable a todos los empleados que han trabajado más de cinco años.
* **Clase EmpleadoPlazaFija:** Hereda de Empleado y añade el cálculo del pago basado en el salario base y las comisiones.
* **Clase EmpleadoPorHoras:** Hereda de Empleado y realiza el cálculo del pago en función de la tarifa por hora y las horas trabajadas.

# Funcionalidades de Python Utilizadas

Se utilizaron las siguientes funcionalidades de Python en el código:

* **Herencia:** Se empleó herencia para reutilizar el código y mantener una estructura clara entre los tipos de empleados. Esto facilita el manejo de distintos cálculos de pago sin duplicar código.
* **Cálculos matemáticos:** Se realizan operaciones simples para calcular el pago total de cada empleado, teniendo en cuenta su tipo de contrato y los años trabajados.
* **Condicionales:** Se utiliza un condicional dentro del método calcular\_bono para determinar si el empleado tiene derecho a un bono adicional.

# Justificación de las Decisiones

Se optó por utilizar POO para reflejar la naturaleza jerárquica de los empleados en la empresa. Al encapsular las propiedades y comportamientos específicos de cada tipo de empleado en sus respectivas clases, se logra un código más limpio y fácil de mantener. La herencia permite que las clases compartan atributos comunes, como el cálculo del bono, sin necesidad de repetir código.

El uso de cálculos simples y condicionales asegura que el programa funcione de manera eficiente y sea fácil de entender. Esto es especialmente útil cuando se trata de manejar diferentes estructuras de pago, como en el caso de empleados por horas versus empleados con plaza fija.