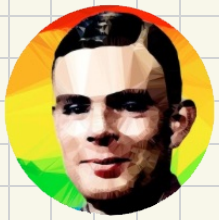
Una Ambiciosa Introducción a Python
Parte II



Una empresa requiere una aplicación para gestionar la liquidación de sueldos de diferentes tipos de empleados. El sistema debe permitir agregar empleados, modificar sus datos y liquidar sus salarios de acuerdo a las reglas correspondientes para cada tipo de empleado.

Tu tarea es modelar esta situación utilizando **clases y objetos**, teniendo en cuenta las diferencias entre los tipos de empleados, así como las deducciones que corresponden a cada uno.

Requerimientos:

- Define una estructura de clases que represente a los empleados de la empresa.
- Implementa la funcionalidad necesaria para liquidar los sueldos, teniendo en cuenta las características específicas de cada empleado.
- Crea una clase gestora que se encargue de gestionar los empleados y realizar las liquidaciones.

Liquidación de Sueldos:

Existen tres tipos de empleados, cada uno con reglas particulares para la liquidación de su salario:

1. **Empleado General**: Tiene un salario base y puede realizar horas extra, que se pagan a un valor adicional. El salario neto se calcula deduciendo un porcentaje del salario bruto más las horas extra.
2. **Gerente**: Además del salario base, puede recibir un bono. Las deducciones se aplican sobre el total de salario más el bono.
3. **Directivo**: Tiene un salario base y puede poseer acciones de la empresa. El salario neto se calcula considerando el valor de las acciones además del salario bruto, aplicando las deducciones correspondientes.

El objetivo es que determines los porcentajes de deducción y las reglas de cada tipo de empleado para calcular su sueldo neto. Asegúrate de utilizar principios como la herencia y el polimorfismo para modelar el sistema de manera eficiente.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

En el desarrollo de software, especialmente en aplicaciones web y de escritorio, es fundamental organizar el código de manera que sea mantenible, escalable y fácil de entender. Para lograr esto, se utilizan diferentes patrones de diseño y arquitecturas. Uno de los patrones mas populares y efectivos es el modelo de capas y, en particular, el modelo **MVC** Modelo-Vista-Controlador.

¿Qué es el Modelo de Capas?

Es un patrón arquitectónico que divide una aplicación en diferentes niveles o capas, cada una con una responsabilidad específica. Esto permite separar preocupaciones, organizar el código de manera modular facilitando la mantenibilidad del sistema.

Capas comunes

1. **Presentación**
   Es la capa que interactúa directamente con el usuario
   Se encarga de presentar información y recibir entradas del usuario

2. **Lógica del Negocio**
   Contiene la lógica principal de la aplicación
   Encargada de procesar datos, aplicar reglas de negocio y tomar decisiones

3. **Acceso a Datos**
   Gestiona la interacción con la base de datos o cualquier otro sistema de almacenamiento
   Encargada de realizar operaciones como consultas, inserciones, actualizaciones y eliminación de datos

Ventajas sobre el modelo de capas

- Separación de preocupaciones: cada nivel tuna una responsabilidad clara, lo que facilita el mantenimiento.

- Reutilización del código: Las capas pueden ser reutilizadas en diferentes partes de la aplicación o en otros proyectos.

- Escalabilidad: Es más fácil escalar la aplicación al poder modificar o mejorar una capa sin afectar a las demás.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Modelo Vista Controlador **MVC**

Es un patrón de diseño que separa una aplicación en tres componentes principales, Modelo, Vista y Controlador. Este patrón es ampliamente utilizado en el desarrollo de aplicaciones web y escritorio.

Modelo (Model)

- Representa los datos y lógica de negocio de la aplicación.
- Encargada de gestionar la información, acceder a la base de datos y aplicar reglas de negocio.

Vista (View)

- Es la interfaz de usuario que muestra la información al mismo.
- Encargada de presentar datos de manera visual y recibir interacción del usuario.

Controlador (Controller)

- Es el intermediario entre el modelo y la vista.
- Recibe las solicitudes del usuario, procesa la información (utilizando el modelo) y decide que vista mostrar.

Esta separación permite que cada parte sea desarrollada y modificada de manera independiente.

Flujo del modelado MVC

1. El usuario realiza una acción.
2. El controlador recibe la solicitud y decide qué hacer.
3. El controlador interactúa con el modelado para obtener o manipular los datos.
4. El modelo devuelve los datos al controlador.
5. El controlador pasa los datos a la vista.
6. La vista muestra la información al usuario.

Una Ambiciosa Introducción a Python
Parte II

Modelo

Controlador

Vista