**TITULO DE LABORATORIO**

**PARCIAL (SALARIO DE UN TRABAJADOR)**

**HANNA KATHERINE ABRIL GÓNGORA**

**KAROL ASLEY ORJUELA MAPE**

**UNIVERSIDAD MANUELA BELTRÁN**

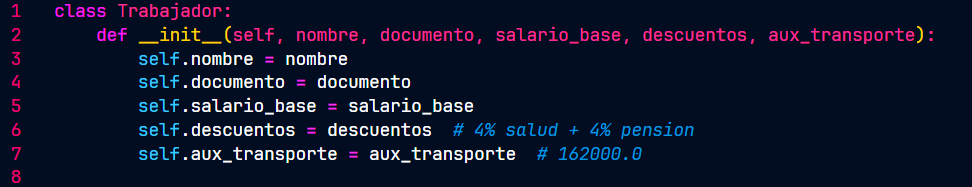
**PROGRAMACIÓN ORIENTADA A OBJECTOS**

**DOCENTE**

**DIANA MARCELA TOQUICA RODRÍGUEZ**

**BOGOTÁ DC VIERNES 22 DE MARZO**

1. **EXPLICACIÓN DEL CÓDIGO EN PYTHON**
   * Se define una clase llamada Trabajador que tiene un método constructor *\_\_init\_\_.* Este método inicializa los atributos de un trabajador, como el nombre, documento, salario base, descuentos en porcentaje total, los cuales incluyen aporte a salud y aporte a pensión, y auxilio de transporte.

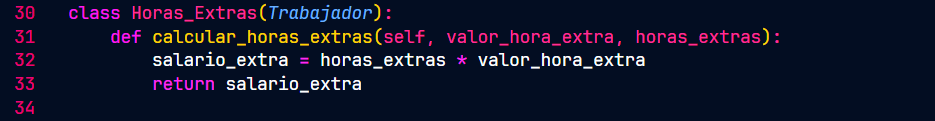


* + El método *calcular\_salario\_total* calcula el salario total de un trabajador teniendo en cuenta todas las prebendas. Toma como parámetros las horas extras trabajadas y el valor por hora extra definido según el tipo de hora extra trabajada. Calcula el salario total sumando el salario base, aplicando un incremento del 10% por ley, agregando el valor de las horas extras y restando los descuentos de salud y pensión (calculados como un porcentaje del salario total). Finalmente, agrega el auxilio de transporte si el salario base es menor o igual a 2,000,000.

Finalmente retorna el salario total, con horas extras y el total en descuentos.



* + Se define una subclase *Horas\_Extras* que hereda de la clase Trabajador. Esta subclase tiene un método *calcular\_horas\_extras* que calcula el salario extra basándose en el valor por hora extra y las horas extras trabajadas que ya están predefinidas en el código.



* + Se define una clase llamada *Registro\_Trabajadores* que tiene una lista para almacenar trabajadores con información como el nombre y el documento de estos además tiene métodos para agregar trabajadores a la lista y listar los trabajadores existentes.

Texto

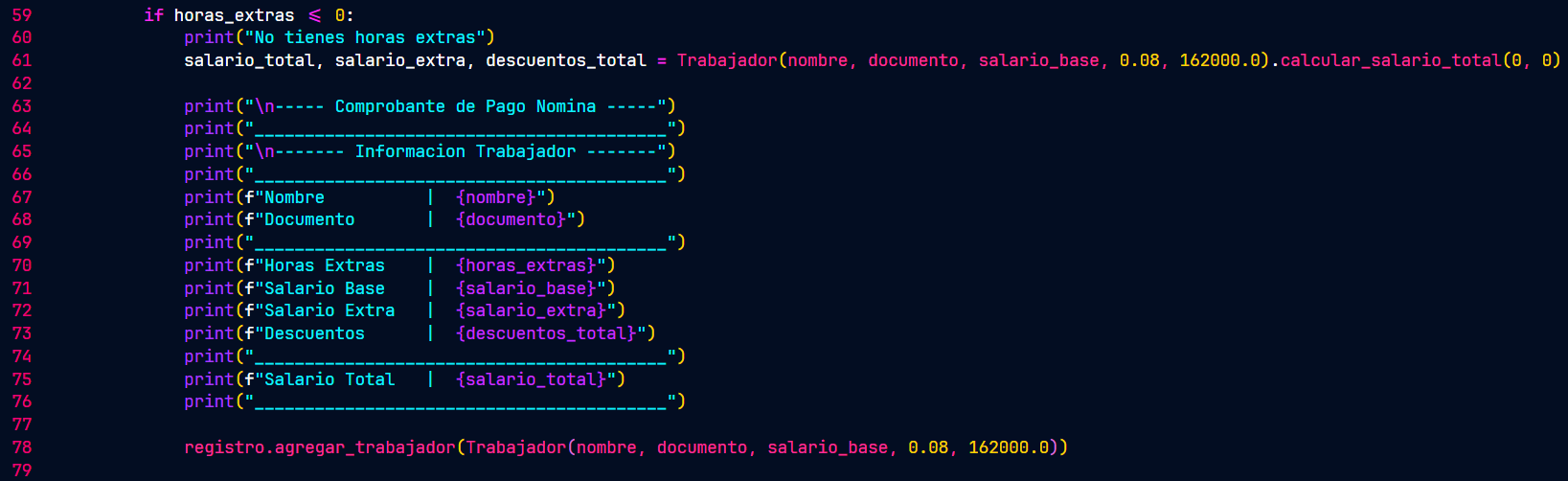
Descripción generada automáticamente

* + Se define la función *main()* que crea un objeto de la clase *Registro\_Trabajadores*. Luego, entra en un bucle donde solicita al usuario ingresar los datos de un trabajador, como nombre, documento, salario base y horas extras trabajadas.

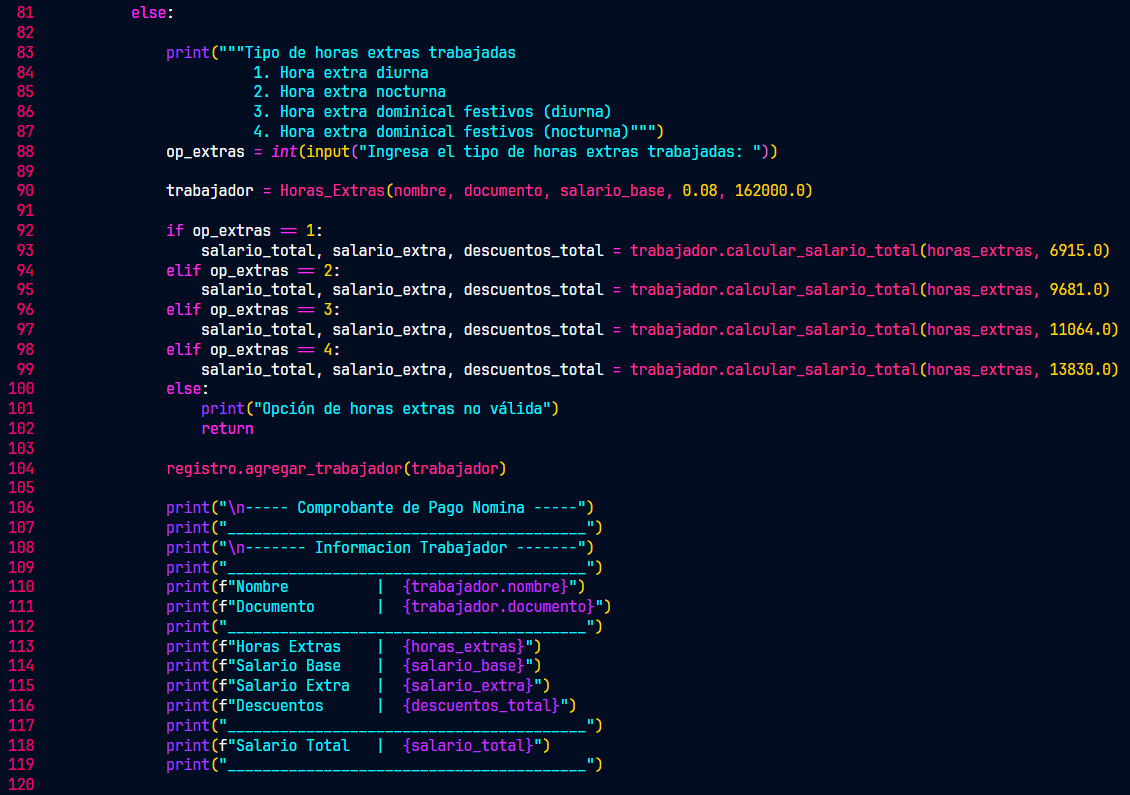
Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

* + Si las horas extras son menores o iguales a cero, el programa asume que el trabajador no tiene horas extras y calcula el salario total sin considerar horas extras y después imprime el comprobante de pago con dichos cálculos realizados y después agrega el trabajador al registro.



* + En caso de que las horas extras sean mayores a cero, el programa entra en un bloque de código para manejar las horas extras y calcular el salario total teniendo en cuenta el tipo de horas extras trabajadas, además se maneja las diferentes opciones de horas extras, calcula el salario total, imprime el comprobante de pago



* + El código restante en la función *main()* ofrece la opción de listar todos los trabajadores registrados.

Texto

Descripción generada automáticamente

* + Finalmente, se ejecuta la función *main()* si el script se ejecuta como el programa principal.

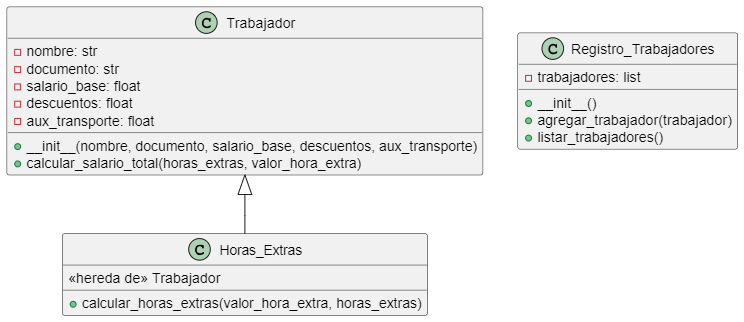


1. **EXPLICACION DEL CODIGO UML**

****

* + **Líneas 1-11:** Se define una clase llamada “Trabajador” con los siguientes atributos:
* nombre: Representa el nombre del trabajador (tipo de dato: cadena de caracteres).
* documento: Representa el número de documento del trabajador (tipo de dato: cadena de caracteres).
* salario\_base: Indica el salario base del trabajador (tipo de dato: número decimal).
* descuentos: Representa los descuentos aplicados al salario (tipo de dato: número decimal).
* aux\_transporte: Es el valor del auxilio de transporte (tipo de dato: número decimal).
* El método \_init\_ (constructor) inicializa los atributos con los valores proporcionados.
* El método calcular\_salario\_total calcula el salario total teniendo en cuenta las horas extras trabajadas y el valor de la hora extra.
  + **Líneas 13-15:** Se define una clase llamada “Horas\_Extras” que hereda de la clase “Trabajador”. Esta clase tiene un método:
* calcular\_horas\_extras: Calcula las horas extras basado en el valor de la hora extra y las horas extras trabajadas.
* Líneas 17-22: Se define una clase llamada “Registro\_Trabajadores” con un atributo:
* lista\_trabajadores: Es una lista que almacena objetos de la clase “Trabajador”.
* Los métodos agregar\_trabajador y listar\_trabajadores permiten agregar nuevos trabajadores a la lista y listar los existentes.
  + **Línea 24:** Indica la relación de herencia entre las clases “Trabajador” y “Horas\_Extras”.
  + **Línea 27:** Marca el final del diagrama UML.

1. **DIAGRAMA DEL CODIGO EN UML**

****

1. **EXPLICACION Y USO**

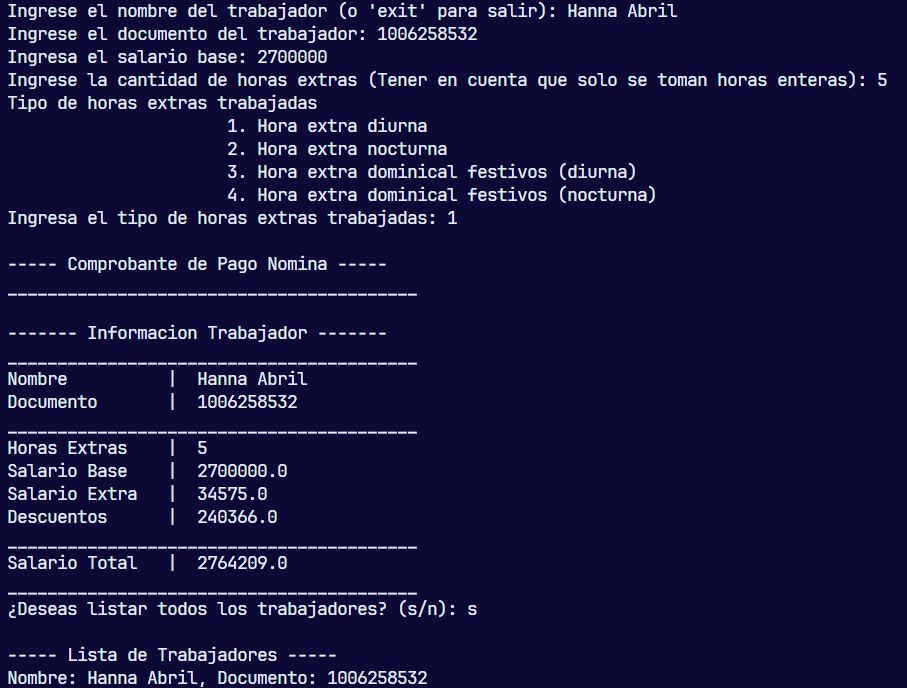
El código proporciona un sistema de registro y cálculo de nómina para trabajadores. Se define una clase llamada Trabajador que tiene atributos como nombre, documento, salario base, descuentos por salud y pensión, y auxilio de transporte. La clase también tiene un método para calcular el salario total, considerando horas extras, incrementos legales, descuentos de salud y pensión además de auxilio de transporte. Además, existe una clase Horas\_Extras que hereda de Trabajador y permite calcular el salario extra por horas extras trabajadas. La clase Registro\_Trabajadores administra una lista de trabajadores y proporciona métodos para agregar trabajadores al registro y listarlos.

La función principal (main) del programa interactúa con el usuario para ingresar información de los trabajadores, como nombre, documento, salario base y horas extras. Dependiendo de si hay horas extras o no, se calcula el salario total y se genera un comprobante de pago con los detalles del trabajador y su salario. También se ofrece la opción de listar todos los trabajadores registrados en el sistema.

En resumen, este código simple facilita la gestión de información y el cálculo de salarios para empleados, brindando una herramienta útil para la administración de nóminas.

1. **DEMOSTRACION DE PRUEBAS Y FUNCIONAMIENTO DEL CODIGO**

**5.1. Prueba 1.**

****

En esta primera prueba se demuestra el funcionamiento con el calculo del salario de un trabajador, en donde se le piden que ingrese la información de esta y que escoja el tipo de horas extras, después de eso se le da el respectivo comprobante de nomina al usuario con toda la información, además de eso se muestra que se agrego el nombre del trabajador a la lista de control.

**5.2. Prueba 2.**

**Texto

Descripción generada automáticamente**

En esta prueba se inscribe otro trabajador con su respectivo salario y sus horas extras en este caso se elije otro tipo de horas extras que tienen otro valor, y se hace el respectivo calculo para después pasar el comprobante de nomina al usuario, e igual que la anterior se ve listado este usuario y el anterior.

**5.3. Prueba 3.**

**Texto

Descripción generada automáticamente**

En esta prueba podemos evidenciar que este trabajador inscrito no realizo horas extras, por lo que en el comprobante de pago se puede evidenciar, que no tiene salario extra y que además no hay horas extras.

**5.4. Prueba 4.**

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

En esta prueba se evidencia el uso de otro tipo de hora extra, e igual que todas las anteriores retorna el comprobante de nómina con la información respectiva.

**5.5. Prueba 5.**

**Texto

Descripción generada automáticamente**

En esta prueba final podemos ver que se inscribió otro usuario y se uso otro tipo de hora extra, e igual que los anteriores dio el comprobante de nómina, pero en este caso lo importante es que se puede evidenciar que se almaceno los nombres de todos los trabajadores anteriores al momento de listarlos y que además de eso el sistema de salida del proceso funciona perfectamente.

1. **DECISIÓN DE DISEÑO O CONSIDERACION IMPORTANTE**

La principal consideración importante para el diseño era saber que parámetros importantes se necesitaban para obtener el salario legal mensual vigente de un trabajador en Colombia, para eso nos referimos a MinTrabajo, para revisar todo el proceso tanto de el salario base como de descuentos legales, además de eso horas extras y auxilios como en este caso de transporte.

Otra de las decisiones importantes que se tomaron era como se iba a mostrar dicha información al usuario, por lo que se decidió por hacer una especie de comprobante de nomina con toda la información importante, incluyendo nombre y el documento del trabajador.

1. **PILARES DE POO IMPLEMENTADOS**

**7.1. Encapsulamiento:**

Se utiliza la encapsulación al definir la clase Trabajador con sus atributos privados (nombre, documento, salario\_base, descuentos, aux\_transporte) estos únicamente serán accedidos desde la misma clase.

**7.2. Abstracción:**

La abstracción se implementa al definir las clases Trabajador, Horas\_Extras, y Registro\_Trabajadores, donde cada una representa un concepto específico del mundo real y proporciona métodos para interactuar con esos conceptos de manera abstracta sin preocuparse por los detalles internos de implementación.

**7.3. Herencia:**

Se utiliza la herencia al definir la clase Horas\_Extras como clase hija de Trabajador. La clase hija hereda los atributos y métodos de la superclase Trabajador, lo que permite reutilizar código.

1. **CONCLUCIONES**

Al realizar el código anterior, se aprendieron varios conceptos importantes relacionados con la programación en Python y el diseño de sistemas básicos de administración de datos y cálculos. Algunas de las conclusiones son:

* Programación Orientada a Objetos (POO): Se utilizó la POO para definir clases como Trabajador, Horas\_Extras y Registro\_Trabajadores, lo cual facilitó la organización y reutilización del código al encapsular atributos y métodos relacionados.
* Herencia y Polimorfismo: Se aplicó el concepto de herencia al crear la clase Horas\_Extras que hereda de Trabajador, lo que permitió reutilizar funcionalidades y métodos. Además, se observó un nivel básico de polimorfismo al utilizar métodos comunes en diferentes clases.
* Interacción con el Usuario: Se implementó la entrada de datos mediante la función input() para interactuar con el usuario y obtener información como nombre, documento y horas extras. Esto es fundamental para crear programas interactivos y amigables para el usuario.
* Cálculos y Lógica de Negocio: Se realizó la lógica de cálculo de salarios teniendo en cuenta el salario base, horas extras, descuentos y otros factores, lo cual es esencial en aplicaciones de gestión de nóminas y finanzas.
* Estructuras de Datos: Se utilizó una lista para almacenar objetos de tipo Trabajador en la clase Registro\_Trabajadores, lo que demostró cómo gestionar y manipular datos en estructuras de datos comunes en Python.

En resumen, al realizar este código se adquirieron habilidades en programación orientada a objetos, manejo de datos, interacción con el usuario y lógica de negocios, aspectos fundamentales para el desarrollo de aplicaciones más complejas y funcionales en Python y otros lenguajes de programación.