EJERCICIO 1

Crear una **clase Coche**, a través de la cual se puedan crear objetos que almacenen el color del coche, la marca, el modelo, el número de caballos, el número de puertas y la matrícula. Crear el metodo __init__, el método __str__. Crear también el método __eq__ que compare cuando dos objetos son iguales **(tienen la misma matricula)**.

EJERCICIO 2

Crea una clase llamada **Libro** que guarde la información acerca de un libro de una biblioteca. La clase debe guardar el título del libro, ISBN, autor, número de ejemplares totales del libro y número de ejemplares prestados. La clase contendrá los siguientes métodos:

- Constructor__init__.
- Método __str__.
- Método __eq__ donde dos libros son iguales si tienen el mismo ISBN.
- Método llamado ***préstamo*** que incremente el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar.
- Método ***devolución*** que produce la devolución de un libro. Si no se ha prestado ningún libro no se puede devolver.

EJERCICIO 3

Crear la clase **Cerveza** que tiene los siguientes atributos:

- Código de identificación (alfanumérico)
- Nombre.
- Tipo (rubia, tostada, roja y negra).
- Elaboración artesanal o no.
- Precio.
- Existencias

Además de los siguientes métodos:

- Constructor con todos los parámetros menos el id que tiene el prefijo CERV-X , donde X es un número que se controla desde la clase y que se incrementa en 1 cada vez que se crea una instancia de la clase Cerveza.
- **servir_cerveza(int)**. Resta de las existencias y si no hay suficientes sirve las que pueda.
- **reponer_cerveza(int)**. Añade existencias.
- Método __**eq**__ donde una cerveza es igual a otra si coinciden los códigos identificativos.
- Método __**str**__.

EJERCICIO 4

Crear la clase **Joya** que tiene los siguientes atributos:

- Código de identificación (alfanumérico)
- Nombre.
- Marca.
- Tipo (collar, anillo, brazalete, diadema o broche).
- Precio.
- Peso en gramos.
- Quilates.

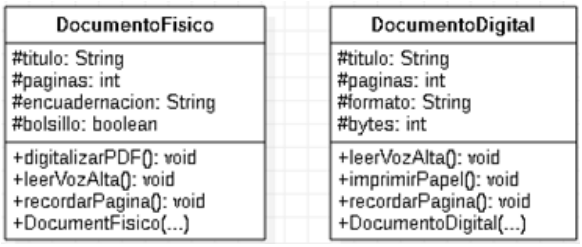
Además de los siguientes métodos:

- Constructor con todos los parámetros menos el id que tiene el prefijo JOYA-X , donde X es un número que se controla desde la clase y que se incrementa en 1 cada vez que se crea una instancia de la clase Joya.
- Método __eq__ donde una joya es igual a otra si coinciden los códigos identificativos.
- Método __str__.
- Método **pureza()** que devuelve un número con la pureza de oro en función del peso y los quilates (1quilate=aprox 4,167% de oro)

EJERCICIO 5

Dado el siguiente diagrama de clases, **escribir clases de código Python refactorizando** para simplificar su mantenimiento **usando herencia**:

- **En las clases incluir únicamente** los atributos y métodos descritos en el diagrama con las superclases y subclases necesarias.
- El **constructor** además del método __str__ y __eq__ (dos documentos son iguales si tienen el mismo título).
- **Los métodos void digitalizarPDF, leerVozAlta, imprimir y recordarPagina** solo sacan por pantalla un mensaje con la acción que realizan.



EJERCICIO 6

Crear la aplicación para la gestión de empleados de una empresa que está compuesta por varias clases:

- La clase **Empleado** tiene cuatro atributos**: nombre, dni** como cadena de texto, **sueldo_base y años en la empresa**. Implementar el método __**init**__**, y** __**eq**__ (mismo dni mismo empleado), además del método **sueldo_neto** que devuelve un float con el sueldo deduciéndole un 15% de impuestos. Debe implementarse también un método __**str**__ que debe mostrar todos los atributos y el **sueldoFinal** debe aparecer dentro del string que devuelve el __**str**__.
- La clase **Vendedor** que hereda de **Empleado** y tiene como atributos propios las ventas totales realizadas por el vendedor y la comisión que se lleva por venta realizada. Hay que redefinir el método _**_str**__ con los nuevos datos y también **sueldo_neto** añadiendo la comisión total por ventas realizadas al cálculo que realiza la clase **Empleado**.
- Por último, la clase **Vendedor** tiene un método llamado **vender** que lanzar un mensaje por pantalla informativo.
- La **clase Empresa** que tiene 2 atributos que son el nombre de la empresa y una lista que está inicialmente vacía. El nombre de la empresa se le pasa al constructor.
- La clase **Empresa** tendrá un método para añadir objetos de las clases anteriores a la lista, llamado **contratar_trabajador**.
- Por último, la clase **Empresa** tendrá un método llamado **imprimir_resumen** que recorre la lista completamente y para cada objeto **(**que puede ser **Empleado o Vendedor)** hace un print de lo que devuelve el método __**str**__ correspondiente, donde además en caso de ser un objeto de la clase **Vendedor** ejecutar también el método vender.

EJERCICIO 7

Crear la clase **Producto** que tiene los atributos nombre, identificador (código alfanumérico) y precio. Tiene que tener constructor, __**str**__ **y** __**eq**__**,** donde un producto es igual a otro cuando tienen el mismo identificador. También tendrá el método **comprar** que tiene como parámetro las unidades a llevarse. El método **comprar** devuelve un **float** con el precio multiplicado por la unidades que se quiere llevar. Si al comprar se lleva más de **50 unidades** el precio final se reduce a la mitad.

Crear la clase **Perecedero** que hereda de **Producto**. Se añade un atributo más que es los días para **caducar**. Hay que hacer el método __**str**__. El método **comprar** es similar al de producto solo que le reduce el precio calculado en la clase **Producto** según los días que le queden para caducar:

- Si le queda un día para caducar, el precio final es la cuarta parte del que devuelve comprar de la clase **Producto**.
- Si le quedan 2 días para caducar, el precio final es la tercera parte.
- Si le quedan 3 días para caducar, el precio final es la mitad.

- Si le quedan 4 o más días no se modifica el precio original.

- Crear la **clase Tienda** como se describe a continuación:
  - La clase tiene 1 atributo que es una lista que está inicialmente vacía.
  - La clase tendrá un método para añadir objetos de las clases anteriores llamado **nuevo_producto** a la lista anteriormente mencionada. El método recibe objetos de la clase Producto y Perecedero.
  - La clase Tienda también tendrá un método **__str__** uniendo todos los datos de todos los productos que haya en la lista (**Producto o Perecedero)**, llamando al método **__str__** correspondiente.