1. Diseñar una clase llamada “Vehículo” con los atributos marca, modelo, año y precio. Crear dos objetos pertenecientes a esa clase e imprimir en pantalla la marca, el modelo y el precio de cada vehículo (mediante __str__).
2. En este ejercicio utilizaremos la misma clase que en el ejercicio anterior y añadiremos un método llamado “nombre_completo” que retorne en una cadena los atributos marca y modelo concatenados y separados por un guión (Seat-Ibiza). Crear dos objetos y probar el método.
3. Diseñar una clase Python llamada “Producto” con los atributos nombre, categoría, precio y cantidad. Diseña en esta clase el método __str__ de forma que retorne todos los atributos en un dato de tipo cadena (str).
   Crear dos productos pertenecientes a esa clase y mostrar todos los datos de aquel producto que tenga mayor precio.
4. Modificar la clase “Producto” del ejercicio anterior añadiendo un método que actualice la cantidad de un producto sumándole un valor pasado como parámetro. Mostrar los datos de un producto antes y después de ser modificada su cantidad.

   En los siguientes ejercicios utilizaremos esta lista de productos:

```
productos = [Producto("tomate", "fruta", 2.3, 100),
             Producto("patata", "verdura", 1.5, 200),
             Producto("cebolla", "verdura", 1.8, 150),
             Producto("manzana", "fruta", 3.2, 50),
             Producto("pera", "fruta", 2.7, 75)]
```

5. Visualizar en pantalla aquellos productos que pertenezcan a la categoría ‘verdura’.
6. Visualizar en pantalla aquellos productos cuyo precio esté entre 1.5 y 2.5 (incluidos).
7. Obtener la media de los precios de los productos de la categoría ‘verdura’.
8. Contar cuántos productos tienen un precio entre 2 y 3 (excluidos).
9. Calcular la media de todos los precios de los productos.
10. Diseñar una clase llamada “Mates” con los **métodos estáticos** que se describen:
    a. mayor. Recibe dos números como argumento y retorna el mayor.
    b. producto. Recibe tres números como argumento y retorna su producto.
    c. potencia. Recibe una base y un exponente como argumentos y retorna la base elevada al exponente.

    Probar los métodos programados.
11. Diseñar la clase “Empleado” con los atributos identificador, nombre, departamento, salario. Tener en cuenta que el salario será privado. Define un método para obtenerlo y otro para modificarlo.
    Programa el método __eq__(), de forma que indique si dos empleados son iguales o no en función de su salario.
    Crear varios empleados, mostrar sus datos y comparar si son iguales o no.
12. Diseñar una aplicación Python que trabaje con objetos de la clase “Partido”. Cada partido tendrá como atributos equipo local, equipo visitante, goles local, goles visitante, campeonato y fecha.

La aplicación consta también de una clase llamada “GestionPartidos” que tendrá como atributo de clase una lista de partidos y los métodos siguientes:

- Filtrar por equipo local. Recibe un equipo local como argumento e imprime todos los partidos de ese equipo actuando como local.
- Ganados del equipo. Recibe un equipo como argumento y retorna cuántos partidos ganó ese equipo, independientemente de si actuó como local o como visitante.
- Mostrar los partidos del año pasado como parámetro.
- Mostrar los partidos de una fecha pasada como parámetro.
- Añadir un nuevo partido a la lista de partidos.
- Cuenta partidos. Retorna el número de partidos de la lista.

Prueba las clases y los métodos creados.

13. Para este ejercicio utilizaremos la lista de productos vista antes. Accederemos al primer objeto de la lista y le añadiremos el atributo ***caducado*** con el valor ***True***. Recorremos la lista y visualizamos el atributo __dict__ de cada objeto. Recuerda que este atributo proporciona información sobre los atributos de un objeto, en forma de diccionario.

14. Programar una agenda de contactos.
    Definir la clase Contacto con los atributos nombre, teléfono y correo del contacto.
    a. Añade el método __str__ para que retorne todos los atributos con el formato: **Nombre - teléfono - correo.**
    b. Programa también __repr__ para que retorne los datos del contacto con el formato: **Contacto(nombre, teléfono, correo).**
    c. Programa el método __eq__ para determinar si dos contactos son iguales. En este caso serán iguales si coinciden todos los valores de sus atributos.

    Programa la clase Agenda. Esta clase tendrá una lista de contactos y los métodos.
    - Buscar contacto por nombre y retornar el contacto.
    - Obtener el teléfono de un contacto. Retornar el teléfono.
    - Obtener el correo de un contacto. Retornar el correo.
    - Cambiar el teléfono de un contacto. Retornar True si se pudo hacer el cambio, False en caso contrario.
    - Cambiar el correo de un contacto. Retornar True si se pudo hacer el cambio, False en caso contrario.
    - Listar todos los contactos.
    - Obtener el número de contactos.

15. Diseña una aplicación Python que gestione una lista de ciudades.
    De cada ciudad se guarda nombre, población, país y continente. Se considera que dos ciudades son iguales si tienen el mismo nombre y país.
    La gestión de ciudades incluirá la creación de una clase que contenga métodos para:
    - Mostrar las ciudades de un continente dado.
    - Mostrar las ciudades con una población mayor que un número dado.

- Retornar el número de ciudades de un país dado.
- Retornar el número de ciudades que contienen una cadena en su nombre.
- Retornar la media de la población de las ciudades de un país.
- Retornar una lista con las ciudades de un país.
- Retornar una lista con las ciudades de un continente.
- Retornar la suma de los habitantes de todas las ciudades.
- Añadir ciudad. Este método recibe un objeto ciudad como parámetro e intenta añadir esa ciudad a la lista. Si la ciudad está en la lista no podrá añadirla y retornará False. En caso de añadirla el retorno será True. Para comprobar si un valor está en una lista se puede utilizar:
  - ***if valor in lista***
  - ***if valor not in lista***

Aquí tienes una lista de ciudades que te pueden interesar.

```
list_cities = [
        Ciudad("Bogotá", 8000000, "Colombia", "América"),
        Ciudad("Lima", 10000000, "Peru", "America"),
        Ciudad("Paris", 5000000, "Francia", "Europa"),
        Ciudad("Berlin", 4000000, "Alemania", "Europa"),
        Ciudad("Tokio", 9000000, "Japón", "Asia"),
        Ciudad("Sydney", 3000000, "Australia", "Oceanía"),
        Ciudad("Johannesburgo", 5000000, "Sudáfrica", "África"),
        Ciudad("Moscú", 10000000, "Rusia", "Europa"),
        Ciudad("Nueva York", 8000000, "Estados Unidos", "América"),
        Ciudad("Sao Paulo", 12000000, "Brasil", "América"),
        Ciudad("Buenos Aires", 15000000, "Argentina", "América"),
        Ciudad("Londres", 9000000, "Reino Unido", "Europa"),
        Ciudad("Roma", 4000000, "Italia", "Europa"),
        Ciudad("Pekín", 20000000, "China", "Asia"),
        Ciudad("Delhi", 15000000, "India", "Asia"),
        Ciudad("El Cairo", 7000000, "Egipto", "África"),
        Ciudad("Ciudad del Cabo", 4000000, "Sudáfrica", "África"),
        Ciudad("Melbourne", 5000000, "Australia", "Oceanía"),
        Ciudad("Auckland", 2000000, "Nueva Zelanda", "Oceanía"),
        Ciudad("Brisbane", 3000000, "Australia", "Oceanía"),
        Ciudad("Madrid", 6000000, "España", "Europa"),
        Ciudad("Lisboa", 3000000, "Portugal", "Europa"),
    ]
```

16. En este ejercicio trabajaremos con una lista de películas.

```
list_movies = [
        Movie("The Shawshank Redemption", 1994, "Frank Darabont", "Tim Robbins, Morgan
Freeman", "Drama", 142, "Castle Rock Entertainment"),
        Movie("The Godfather", 1972, "Francis Ford Coppola", "Marlon Brando, Al Pacino",
"Drama", 175, "Paramount Pictures"),
```

```
        Movie("The Dark Knight", 2008, "Christopher Nolan", "Christian Bale, Heath Ledger",
"Action", 152, "Warner Bros. Pictures"),
        Movie("The Lord of the Rings: The Return of the King", 2003, "Peter Jackson", "Elijah
Wood, Viggo Mortensen", "Adventure", 201, "New Line Cinema"),
        Movie("Pulp Fiction", 1994, "Quentin Tarantino", "John Travolta, Uma Thurman",
"Crime", 154, "Miramax Films"),
        Movie("Forrest Gump", 1994, "Robert Zemeckis", "Tom Hanks, Robin Wright", "Drama",
142, "Paramount Pictures"),
        Movie("Inception", 2010, "Christopher Nolan", "Leonardo DiCaprio, Joseph
Gordon-Levitt", "Action", 148, "Warner Bros. Pictures"),
        Movie("The Matrix", 1999, "Lana Wachowski, Lilly Wachowski", "Keanu Reeves, Laurence
Fishburne", "Action", 136, "Warner Bros. Pictures"),
        Movie("The Silence of the Lambs", 1991, "Jonathan Demme", "Jodie Foster, Anthony
Hopkins", "Crime", 118, "Orion Pictures"),
        Movie("The Departed", 2006, "Martin Scorsese", "Leonardo DiCaprio, Matt Damon",
"Crime", 151, "Warner Bros. Pictures"),
        Movie("The Prestige", 2006, "Christopher Nolan", "Christian Bale, Hugh Jackman",
"Drama", 130, "Warner Bros. Pictures"),
        Movie("The Green Mile", 1999, "Frank Darabont", "Tom Hanks, Michael Clarke Duncan",
"Drama", 189, "Castle Rock Entertainment"),
        Movie("The Godfather: Part II", 1974, "Francis Ford Coppola", "Al Pacino, Robert De
Niro", "Drama", 202, "Paramount Pictures"),
        Movie("The Lord of the Rings: The Fellowship of the Ring", 2001, "Peter Jackson",
"Elijah Wood, Ian McKellen", "Adventure", 178, "New Line Cinema"),
        Movie("The Lord of the Rings: The Two Towers", 2002, "Peter Jackson", "Elijah Wood,
Viggo Mortensen", "Adventure", 179, "New Line Cinema"),
        Movie("The Dark Knight Rises", 2012, "Christopher Nolan", "Christian Bale, Tom Hardy",
"Action", 164, "Warner Bros. Pictures"),
        Movie("The Lord of the Rings: The Two Towers", 2002, "Peter Jackson", "Elijah Wood,
Viggo Mortensen", "Adventure", 179, "New Line Cinema"),
        Movie("One Flew Over the Cuckoo's Nest", 1975, "Milos Forman", "Jack Nicholson, Louise
Fletcher", "Drama", 133, "Fantasy Films"),
        Movie("Goodfellas", 1990, "Martin Scorsese", "Robert De Niro, Ray Liotta", "Crime",
146, "Warner Bros. Pictures")
    ]
```

Cada película (clase Movie) tiene los atributos: título, año, director, reparto, género, minutos y productora.

Se trata de programar métodos que permitan:

- Retornar las pelis de una productora pasada como parámetro.
- Retornar las pelis con una cadena en el título. Esta cadena se pasará como parámetro.
- Retornar cuántas pelis superan la duración media.
- Ordenar por el campo duración. Para realizar esta tarea debemos utilizar ***lista.sort()*** y añadir a la clase Movie el método **__lt__** (less than) y en este programar cuando consideramos que una peli es menor que otra.
    - ***def __lt__(self, otra):***
      return self.minutes < otra.minutes

17. Debemos diseñar una aplicación Python que permita gestionar una lista de vehículos.
    a. Cada vehículo tiene como atributos: matrícula, marca, modelo, color, año, kilómetros, potencia y una lista de fechas en las que ha sido reparado.
    b. Dos vehículos se consideran iguales si tienen la misma matrícula.
    c. El constructor no recibe la lista de fechas como parámetro, sino que existe un método llamado ***agregar_reparación*** que añade una fecha a la lista.

    Los métodos asociados a la lista de vehículos son:
    - Añadir vehículo. No pueden existir varios vehículos con idéntica matrícula.
    - Retornar el número de reparaciones de un vehículo, a partir de su matrícula.
    - Eliminar un vehículo de la lista conociendo su matrícula.
    - Ordenar la lista de vehículos por año de compra.
    - Añadir reparación a un vehículo. Recibe matrícula y fecha de reparación.
    - Mostrar todos los vehículos.

    Os dejo unos datos de prueba.

```
lista_vehiculos = [
        Vehiculo("ABC123", "Toyota", "Corolla", "Rojo", 2019, 20000, 150),
        Vehiculo("DEF456", "Nissan", "Sentra", "Azul", 2018, 18000, 140),
        Vehiculo("GHI789", "Chevrolet", "Spark", "Blanco", 2017, 160400, 130),
        Vehiculo("JKL012", "Mazda", "3", "Negro", 2016, 15000, 120),
        Vehiculo("KKL122", "Volkswagen", "Golf", "Blanco", 2020, 230400, 120),
        Vehiculo("MMG122", "Nissan", "Micra", "Azul", 2020, 123000, 86),
        Vehiculo("ZZE123", "Seat", "Ibiza", "Blanco", 2010, 67000, 120),

    ]
```