1. Diseñar una aplicación Python que trabaje con objetos de la clase "VehiculoAlguilado". Cada vehículo alquilado consta de una instancia de "Vehículo", una instancia de "Cliente", una fecha de inicio de alquiler y un número de días.

    - Un vehículo tiene una matrícula, marca, modelo, potencia y precio de alquiler.
    - De cada cliente se debe guardar un identificador, nombre, apellidos y correo electrónico.
    - **Tanto clientes como vehículos tendrán existencia propia.**
    - Añade a la clase "VehículoAlquilado" el método "importe_alquiler". Este método retorna el resultado de multiplicar el precio de alquiler por el número de días de alquiler.

    Prueba las clases que has diseñado de la siguiente forma.
    - Crea dos vehículos y dos clientes.
    - A partir de esos objetos crea dos instancias de la clase "VehiculoAlquilado". Para cada una de ellas muestra el importe del alquiler.

2. Diseñar una aplicación Python que sirva para programar el juego del "Amigo Invisible". La aplicación debe trabajar con las clases "Persona" y "AmigoInvisible". **Las Personas solo deben existir dentro de la clase AmigoInvisible (Delegación).**
    a. La clase Persona tiene como atributos nombre de persona y correo electrónico. **No podrán repetirse personas con la misma dirección de correo o nombre.**
    b. La clase AmigoInvisible tendrá las siguientes características:
        i. Atributos: descripción, fecha del evento y lista de personas participantes.
        ii. Métodos.
            1. Añadir persona. Recibe nombre y correo electrónico y añade la persona al juego. Retorna True o False según se pudo añadir o no.
            2. Mostrar personas. Muestra en pantalla los datos de los participantes en el juego.
            3. Sortear. Empareja a los participantes y retorna un diccionario con las parejas creadas **(solo los nombres).**

    Prueba las clases creando un juego, añade personas al mismo y realiza el sorteo. Por último imprime el diccionario resultante.

3. Herencia. Diseñar una aplicación Python que permita trabajar con un conjunto de publicaciones tal y como se detalla.
    a. De cada **publicación** se guarda título, editorial, fecha, idioma.
    b. Las publicaciones pueden ser libros, revistas y periódicos.
    c. De los **libros** se guarda título, editorial, fecha, idioma, isbn, autor y género.
    d. De las **revistas** título, editorial, fecha, idioma, número y área.
    e. De los **periódicos** título, editorial, fecha, idioma y sección.

Crea una clase llamada ListaPublicaciones con un atributo de clase que contenga una lista de publicaciones con valores de prueba y programa en ella los siguientes métodos:

- Mostrar las publicaciones que sean del tipo Libro.
- Retornar el número de publicaciones de un tipo que se pasará como parámetro.
- Retornar una lista con las instancias de Revista.
- Retornar un diccionario con las parejas tipo-número de publicaciones. Con los datos de partida el diccionario debe coincidir con este.
  {'Libro': 2, 'Revista': 5, 'Periódico': 6}
- Retornar una tupla con los idiomas sin repeticiones.

Os dejo unos datos de partida. Ojo, he colocado el ISBN del libro como primer argumento en el constructor de Libro.

```
publicaciones = [
        Libro(
            "1234",
            "El Quijote",
            "Anaya",
            date(1605, 1, 1),
            "Español",
            "Cervantes",
            "Novela",
        ),
        Libro(
            "5678",
            "El Señor de los Anillos",
            "Minotauro",
            date(1954, 7, 29),
            "Inglés",
            "J.R.R. Tolkien",
            "Fantasía",
        ),
        Revista("Muy Interesante", "G+J", date(1981, 1, 1), "Español", 123, "Ciencia"),
        Revista(
            "National Geographic", "RBA", date(2022, 1, 1), "Inglés", 123, "Naturaleza"
        ),
        Revista("Vogue", "Conde Nast", date(1892, 1, 1), "Inglés", 123, "Moda"),
        Revista("Time", "Time Inc.", date(2024, 1, 1), "Inglés", 123, "Actualidad"),
        Revista("Hola", "Hola S.L.", date(2024, 1, 1), "Español", 123, "Corazón"),
        Periodico("El País", "Prisa", date(1999, 1, 1), "Español", "Internacional"),
        Periodico(
            "The New York Times",
            "The New York Times Company",
            date(1851, 1, 1),
            "Inglés",
            "Internacional",
        ),
        Periodico(
            "The Guardian",
            "Guardian Media Group",
            date(1821, 1, 1),
            "Inglés",
            "Internacional",
```

```
        ),
        Periodico(
            "Le Monde", "Groupe Le Monde", date(1944, 1, 1), "Francés", "Internacional"
        ),
        Periodico(
            "La Repubblica",
            "GEDI Gruppo Editoriale",
            date(1976, 1, 1),
            "Italiano",
            "Internacional",
        ),
        Periodico(
            "Bild", "Axel Springer SE", date(1952, 1, 1), "Alemán", "Internacional"
        ),
    ]
```

4. Diseñar una aplicación Python que sirva para proporcionar información sobre los cines de una determinada ciudad.
    - De cada cine se almacena un id, nombre, dirección, teléfono y correo electrónico. No pueden repetirse cines con la misma id.
    - Cada cine estará formado por una o varias salas. Por defecto todos los cines tendrán las salas 1, 2 y 3.
    - De cada sala se guarda un número, capacidad y si permite o no las reproducciones 3D.

    La aplicación debe permitir:
    - Añadir nuevo cine.
    - Añadir salas a un cine. El número de sala no puede repetirse dentro de un mismo cine. Para crear las salas se utilizará “delegación”.
    - Retornar una lista con los nombres de los cines.
    - Retornar un diccionario con las parejas nombre-cine:teléfono.
    - Retornar un diccionario con las parejas nombre-cine:número-de-salas.
    - Retornar la capacidad total de un cine conociendo su id. La capacidad de un cine será la suma de las capacidades de sus salas.

```
lista_cines = [
        Cine(1, "Cinepolis", "Av. Warner Sur 123", "924334949", "cinepolis@cinepolis.es"),
        Cine(2, "Cine y palomitas", "Av. Hitchcock 456", "924004949", "cinepalomitas@cine.es"),
        Cine(3, "Cine Tonalá", "Tonalá 123", "924004949", "tonala@cines.es"),
        Cine(4, "Ramoncine", "Ramoncín 123", "924000001", "ramoncine@cines.es"),
        ]
```

5. Clases Abstractas. Diseñar una aplicación que gestione una serie de vehículos.
    - De cada vehículo se guarda matrícula, marca, modelo y precio. **No podrán instanciarse objetos de esta clase.**
    - Los vehículos pueden ser turismos, furgonetas y autobuses.

- De los turismos se guarda además, número de puertas y tipo de combustible.
- De las furgonetas la carga que permiten.
- De los autobuses el número de asientos.
- La clase base Vehículo debe tener un método para incrementar el precio del vehículo en una cantidad pasada como argumento.

La aplicación debe permitir:

- Guardar los vehículos en un diccionario. Las parejas del diccionario serán matricula-vehículo (turismo, furgoneta o autobús).
- Añade métodos para:
  - Añadir un vehículo al diccionario.
  - Buscar un vehículo por su matrícula y retornarlo.
  - Modificar la carga de una instancia de furgoneta. Recibe la matrícula y la nueva carga y retorna True si se pudo hacer el cambio o False en caso contrario.
  - Mostrar todos los vehículos (método str de cada tipo).
  - Retornar una lista de marcas de vehículos sin repeticiones.
  - Retornar una tupla con los objetos de un tipo concreto pasado como argumento.

6. En este ejercicio trabajaremos con **una serie de números enteros** y desarrollaremos métodos que trabajen con los mismos.
   Para ello se creará:
   a. Una interfaz llamada IGestionNumeros que tenga los métodos abstractos: nuevo_numero, buscar_numero (retorna True o False según el número esté o no en la serie) , ordenar y mostrar.
   b. Una clase llamada ListaNumeros que implemente esa interfaz y que trabaje con una lista (list) de números. Esta lista de números se pasará como parámetro en el constructor.
   c. Una clase llamada TuplaNumeros que implemente esa interfaz y que guarde los números en una tupla. La tupla se pasará al constructor como argumento.

7. Crear una clase llamada Pedido con los atributos id, fecha del pedido, cliente y lista de productos (solo el nombre). Siendo ***pedido*** una instancia de la clase ***Pedido***, añade métodos que permitan las siguientes operaciones.
   a. pedido + “patatas”. Añadir las patatas a la lista de productos.
   b. pedido - “lentejas”. Quitar las lentejas de la lista de pedidos.
   c. len(pedido). Obtener el número de productos del objeto pedido.
   d. pedido += pedido_dos. Añadir la lista de productos del pedido pedido_dos a pedido.
   e. bool(pedido). Retorna True si el objeto pedido contiene algún elemento en la lista de productos.

8. Diseñar una aplicación Python que sirva para realizar un sorteo. Para ello, se dispondrá de una serie de premios y de un conjunto de participantes. La aplicación debe asignar o sortear los premios entre los participantes y generar un listado del tipo **"Premio MMM -> Participante NNN"** o similar.

   La aplicación debe tener un menú con las siguientes opciones.
   - 1. Añadir participantes.
   - 2. Añadir premios.
   - 3. Eliminar participantes.
   - 4. Eliminar premio.
   - 5. Listar participantes.
   - 6. Listar premios.
   - 7. Sortear los premios entre los participantes.
   - 8. Finalizar.

   Sólo se podrá llevar a cabo el sorteo si existen premios y participantes.
   El número de participantes debe ser mayor que el número de premios.
   Un participante sólo podrá recibir un premio.
   No pueden quedar premios sin asignar.
   Es conveniente empezar la aplicación con un conjunto de datos de partida.