La base de datos que usaremos para estos ejercicios será “almacén”.

1. Diseñar una aplicación Python que permita listar los clientes de la base de datos.
2. Diseñar una aplicación Python que muestre los productos (código, nombre y categoría) de una categoría que será pasada como argumento. Crea una función que retorne los productos por categoría.
3. Diseñar una aplicación Python que actualice las unidades de un producto mediante una función. Se pasarán el código de producto y las unidades a sumar, como argumentos.
4. Diseñar una aplicación Python que elimine una categoría de la tabla categoriasProductos.

5. Diseñar una aplicación Python con las siguientes características.
    a. Debe contener una clase que retorne una conexión a la base de datos almacén siguiendo el patrón de diseño Singleton.
    b. Debe contener una clase con métodos para:
        i. Retornar una lista con código de producto, nombre y precio de compra de los productos de una categoría pasada como parámetro.
        ii. Actualizar el precio de los productos de una categoría sumando o restando un porcentaje a ese precio. La categoría y el porcentaje deben ser pasados como argumentos. Retornar el número de filas actualizadas.

6. Diseñar una aplicación Python que permita volcar en un archivo PDF los datos de la tabla clientes de la base de datos almacén. Las columnas que deben aparecer son: número de cliente, apellido y nombre concatenados, teléfono y correo. Los datos deben mostrarse ordenados por apellido y nombre.
   Para generar archivos pdf se utiliza la librería ***reportlab***.

7. Diseñar una aplicación Python que permita añadir a la tabla clientes de la base de datos almacén, los datos de los clientes existentes en el archivo ***clientes.csv***. La aplicación debe tener las siguientes características:
    a. Una clase que retorne la conexión a la base de datos y que cumpla el patrón de diseño Singleton.
    b. Una clase con:
        i. Un método que permita leer el archivo csv y retornar sus datos.
        ii. Un método capaz de añadir los clientes a la base de datos y retornar el número de clientes que se han añadido.
    c. Si se produce algún error al añadir un cliente (por ejemplo, duplicar una PK), el proceso debe continuar.

8. Los ejercicios que siguen están pensados para trabajar con el ORM de Django. Por tanto para realizarlos debes:

- Crear un proyecto Django al que llamaremos ***Ejercicio08_Empleados_ORM.***
- Crear dentro del proyecto una aplicación llamada ***empleados.***
- Registrar la app en el archivo ***settings.py.***

- La base de datos. Trabajaremos con SQLite y la base de datos la llamaremos **empleados.db.**
- Abrir el archivo models.py y crear el modelo Empleado. Estudia la imagen siguiente.

```
class Empleado(models.Model):
    nombre = models.TextField(max_length=50)
    apellidos = models.TextField(max_length=50)
    salario = models.DecimalField(decimal_places=4, max_digits=12)
    departamento = models.TextField(max_length=50, blank=True, null=True)
    fecha_contratacion = models.DateField(null=True, blank=True, auto_now_add=True)


    def __str__(self):
        return f"{self.nombre} {self.apellidos} - {self.salario} - {self.departamento}"
```

- Haz las migraciones para crear la tabla en la base de datos.
- Diseña los siguientes scripts. Para que estos scripts sean funcionales deben incluir al principio de los mismos las siguientes líneas.

```
import os

import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Ejercicio06_Empleados_ORM.settings')
django.setup()
```

  - Script 1. Insertar varios empleados.
  - Script 2. Listar los empleados.
  - Script 3. Listar los empleados del departamento de Ventas.
  - Script 4. Buscar un empleado por su id.
  - Script 5. Listar los empleados ordenados por apellidos y nombre.
  - Script 6. Listar los empleados que no pertenecen al departamento de Ventas.
  - Script 7. Listar los empleados con la cadena 'cía' en el atributo apellidos.
  - Script 8. Listar los empleados del departamento de Ventas con salario superior a 2000 euros.
  - Script 9. Listar los tres empleados con mayor salario.
  - Script 10. Eliminar un empleado localizado por su id.
  - Script 11. Actualizar el salario de un empleado localizado por su id.
  - Script 12. Calcular salario mínimo, máximo, medio y contar los empleados del departamento de ventas.

9. Seguimos trabajando con los modelos de Django. Crearemos ahora una nueva aplicación Django con las siguientes características:
   - Nombre del proyecto: Ejercicio09_CursosFormacion.
   - Nombre de la app: cursos.
   - Base de datos sqlite con el nombre: cursos.db.
   - Modelo.

```python
class Curso(models.Model):

    nombre = models.CharField(max_length=100)
    descripcion = models.TextField()
    horas = models.IntegerField()
    tipo = models.CharField(max_length=50)
    fecha_inicio = models.DateField()
    fecha_fin = models.DateField()
    precio = models.DecimalField(max_digits=10, decimal_places=2)

    class Meta:
        db_table = 'cursos'
```

- Scripts.
    1. Insertar varios cursos en la base de datos.
    2. Filtrar los cursos por tipo (presencial, online o semipresencial).
    3. Obtener todos los tipos de cursos.
    4. Contar los cursos por tipos.
    5. Mostrar los que no están iniciados.
    6. Mostrar los que tienen una duración entre dos valores.
    7. Actualizar el precio de todos los cursos presenciales, sumarle 50 euros a cada uno.
    8. Cursos que están desarrollándose.
    9. Crear un PDF con los cursos próximos. Aquellos que no han comenzado.

10. Crear el siguiente proyecto Django y los scripts asociados.
    - Nombre del proyecto: GestionTareas.
    - Nombre de la app: tareas.
    - Base de datos sqlite: tareas.db.
    - Modelo.

```python
class Tarea(models.Model):
    titulo = models.CharField(max_length=100)
    descripcion = models.TextField()
    fecha_creacion = models.DateTimeField(auto_now_add=True)
    fecha_vencimiento = models.DateTimeField()
    fecha_finalizacion = models.DateTimeField(null=True, blank=True)
    completada = models.BooleanField(default=False)

    def __str__(self):
        return self.titulo

    class Meta:
        db_table = "tareas"
```

- Scripts.
    1. Insertar varias tareas.
    2. Mostrar las completadas.
    3. Mostrar las tareas de un año.
    4. Contar las tareas por año.
    5. Actualizar las tareas de un año a completadas.
    6. De las tareas completadas, mostrar la diferencia de días entre su creación y finalización.