



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO



**Tecnológico Nacional de México:** Campus Tuxtla Gutiérrez

## **Ingeniería en Sistemas Computacionales**

**Actividad:** Jerarquía de clases de Gestión de  
Biblioteca

### **Tópicos Avanzados**

**Prof.:** Luis Alberto Ríos Coutiño

Vázquez Guzmán Miriam

**Lugar:** Tuxtla Gutiérrez, Chiapas

**Fecha:** 14 de mayo del 2025

## Introducción

En el presente ejercicio se ha desarrollado un sistema básico de gestión de biblioteca utilizando el lenguaje de programación Python. El objetivo principal fue aplicar los conceptos fundamentales de la programación orientada a objetos (POO), tales como herencia, encapsulamiento y polimorfismo, a través de una jerarquía de clases que simula el comportamiento de libros, revistas y usuarios de una biblioteca. Este tipo de sistemas es ampliamente utilizado en entornos reales donde se requiere controlar el préstamo y devolución de materiales bibliográficos

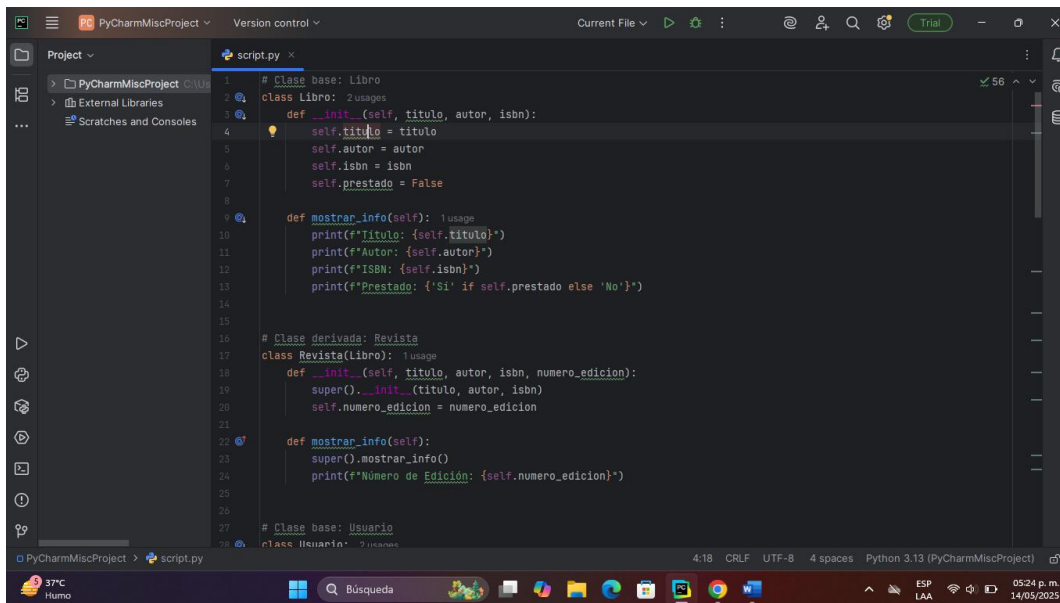
## Desarrollo

Para la construcción del sistema, se definieron las siguientes clases:

- **Libro:** clase base que contiene atributos comunes como título, autor e ISBN, además de un método para mostrar la información del libro. También se incorpora un atributo adicional para verificar si el libro está prestado.
- **Revista:** clase hija que hereda de Libro y añade un atributo exclusivo: el número de edición. Esta clase también redefine el método para mostrar información, incluyendo los datos heredados y el número de edición.
- **Usuario:** clase abstracta que representa a cualquier persona que pueda interactuar con la biblioteca. Posee métodos para prestar y devolver libros, los cuales son implementados de forma específica por las clases hijas.
- **Estudiante y Profesor:** subclases de Usuario que representan distintos tipos de usuarios. Se aplica el principio de **polimorfismo**, permitiendo que el método prestar se comporte de forma distinta según el tipo de usuario. Por ejemplo, un estudiante puede prestar hasta 3 libros, mientras que un profesor puede tener hasta 5 libros prestados simultáneamente.

Figura 1

Paso 1

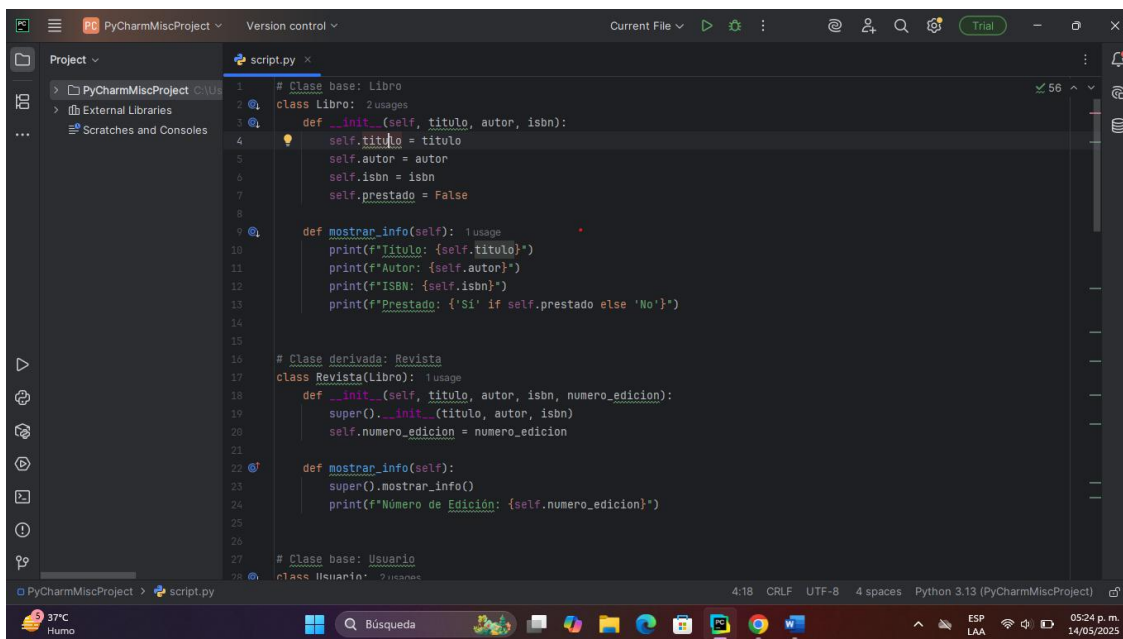


The screenshot shows the PyCharm IDE interface. The 'Project' view on the left shows the 'PyCharmMiscProject' folder. The 'script.py' file is open in the editor. The code defines a base class 'Libro' and a derived class 'Revista'. The 'Libro' class has attributes 'titulo', 'autor', 'isbn', and 'prestado', and a method 'mostrar\_info'. The 'Revista' class inherits from 'Libro' and adds the attribute 'numero\_edicion' and a method 'mostrar\_info' that calls the parent's method and prints the edition number. The status bar at the bottom shows '4:18 CRLF UTF-8 4 spaces Python 3.13 (PyCharmMiscProject)'.

```
1 # Clase base: Libro
2 class Libro: 2 usages
3     def __init__(self, titulo, autor, isbn):
4         self.titulo = titulo
5         self.autor = autor
6         self.isbn = isbn
7         self.prestado = False
8
9     def mostrar_info(self): 1 usage
10        print(f"Titulo: {self.titulo}")
11        print(f"Autor: {self.autor}")
12        print(f"ISBN: {self.isbn}")
13        print(f"Prestado: {'Si' if self.prestado else 'No'}")
14
15
16 # Clase derivada: Revista
17 class Revista(Libro): 1 usage
18     def __init__(self, titulo, autor, isbn, numero_edicion):
19         super().__init__(titulo, autor, isbn)
20         self.numero_edicion = numero_edicion
21
22     def mostrar_info(self):
23         super().mostrar_info()
24         print(f"Número de Edición: {self.numero_edicion}")
25
26
27 # Clase base: Usuario
28 class Usuario: 2 usages
```

Figura 2

Paso 2



The screenshot shows the same PyCharm IDE interface as Figure 1. The 'script.py' file is open. A red dot is visible next to the 'def mostrar\_info(self):' line in the 'Libro' class, indicating a syntax error. The code is identical to the one in Figure 1. The status bar at the bottom shows '4:18 CRLF UTF-8 4 spaces Python 3.13 (PyCharmMiscProject)'.

```
1 # Clase base: Libro
2 class Libro: 2 usages
3     def __init__(self, titulo, autor, isbn):
4         self.titulo = titulo
5         self.autor = autor
6         self.isbn = isbn
7         self.prestado = False
8
9     def mostrar_info(self): 1 usage
10        print(f"Titulo: {self.titulo}")
11        print(f"Autor: {self.autor}")
12        print(f"ISBN: {self.isbn}")
13        print(f"Prestado: {'Si' if self.prestado else 'No'}")
14
15
16 # Clase derivada: Revista
17 class Revista(Libro): 1 usage
18     def __init__(self, titulo, autor, isbn, numero_edicion):
19         super().__init__(titulo, autor, isbn)
20         self.numero_edicion = numero_edicion
21
22     def mostrar_info(self):
23         super().mostrar_info()
24         print(f"Número de Edición: {self.numero_edicion}")
25
26
27 # Clase base: Usuario
28 class Usuario: 2 usages
```

Figura 3

Paso 3

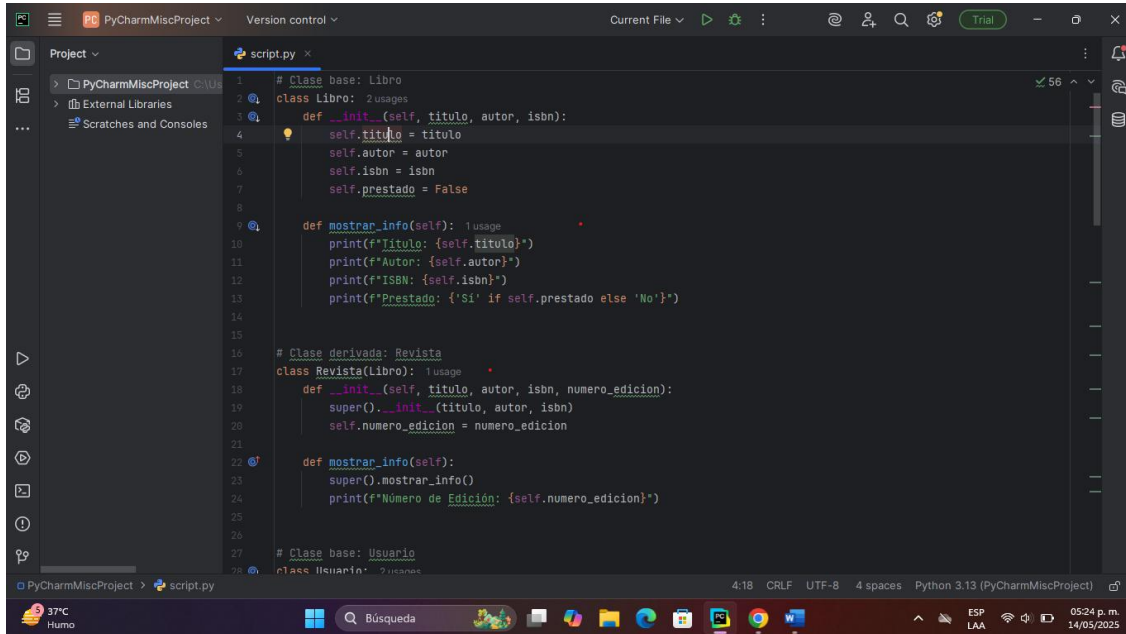


Figura 4

Paso 4

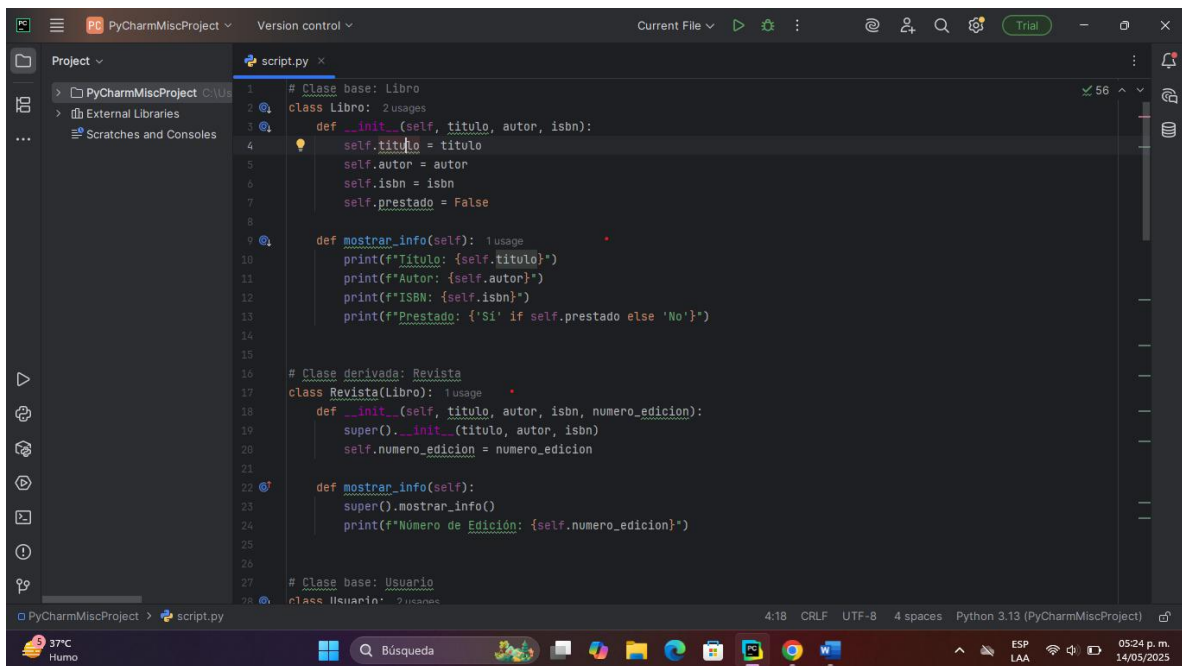
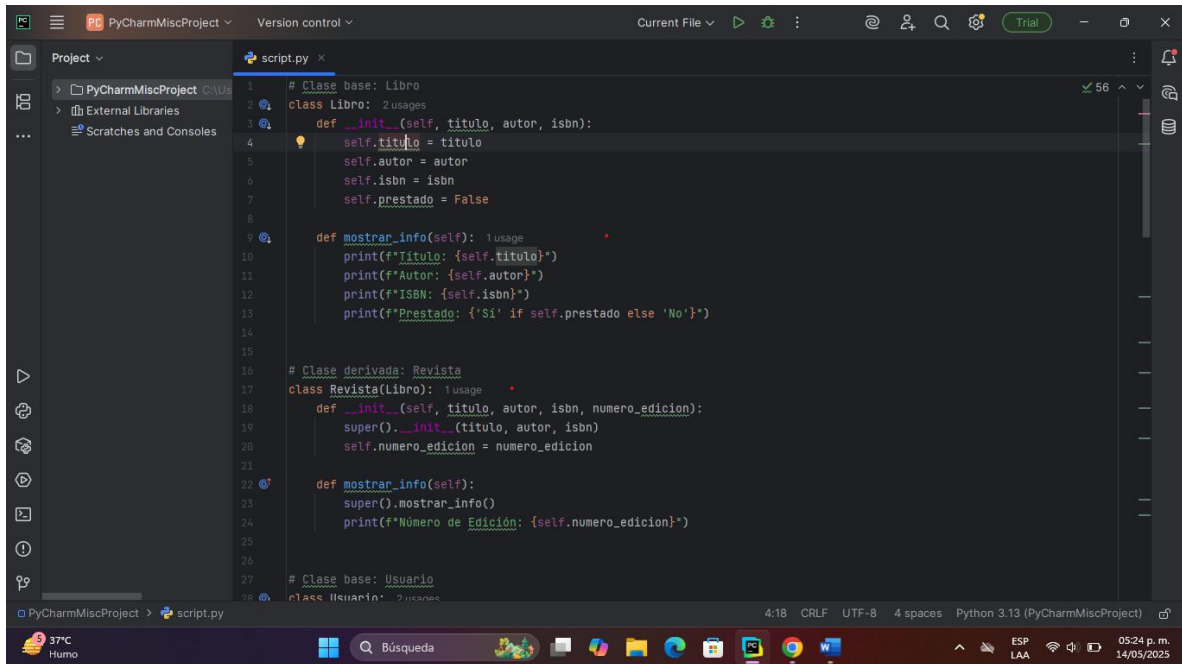


Figura 5

Paso 5



```
1 # Clase base: Libro
2 class Libro:
3     def __init__(self, titulo, autor, isbn):
4         self.titulo = titulo
5         self.autor = autor
6         self.isbn = isbn
7         self.prestado = False
8
9     def mostrar_info(self):
10         print(f'Título: {self.titulo}')
11         print(f'Autor: {self.autor}')
12         print(f'ISBN: {self.isbn}')
13         print(f'Prestado: {'Si' if self.prestado else 'No'}')
14
15 # Clase derivada: Revista
16 class Revista(Libro):
17     def __init__(self, titulo, autor, isbn, numero_edicion):
18         super().__init__(titulo, autor, isbn)
19         self.numero_edicion = numero_edicion
20
21     def mostrar_info(self):
22         super().mostrar_info()
23         print(f'Número de Edición: {self.numero_edicion}')
24
25 # Clase base: Usuario
26 class Usuario:
```

## **Conclusión**

Este ejercicio ha permitido demostrar la aplicación práctica de los conceptos de programación orientada a objetos en Python, mediante el diseño de un sistema simple pero funcional para la gestión de préstamos en una biblioteca. Se evidenció cómo la herencia facilita la reutilización de código, el polimorfismo permite la personalización del comportamiento de los métodos y el encapsulamiento garantiza el control del estado de los objetos. Este tipo de soluciones constituye una base sólida para proyectos más complejos, como sistemas integrales de gestión bibliotecaria con bases de datos o interfaces gráficas.