



[LCD] MÓDULO 5

Bienvenido al **Módulo 5** del curso **Lenguaje de Ciencia de Datos I.**

INTRODUCCIÓN

Estimado estudiante:

En el presente módulo comprenderás los fundamentos de la Programación Orientada a Objetos (POO) y su aplicación en la ciencia de datos. Explorarás los principios de clases, objetos, encapsulación, herencia y polimorfismo, analizando cómo cada uno contribuye a estructurar, reutilizar y escalar el código en el desarrollo de modelos y algoritmos.

Te deseamos una excelente experiencia de aprendizaje en este curso.

¡Muchos éxitos!

 TEMA 5: PROGRAMACIÓN ORIENTADA A OBJETOS

 CIERRE

TEMA 5: PROGRAMACIÓN ORIENTADA A OBJETOS

PROGRAMACIÓN ORIENTADA A OBJETOS

Llegó el momento para que te conviertas en el protagonista de tu aprendizaje.

- Descubre y analiza la información que se presenta en la lección interactiva para transformarla en nuevo conocimiento.





La POO transforma el código en un sistema ordenado y eficiente, capaz de crecer y adaptarse a las necesidades de la ciencia de datos moderna.

La Programación Orientada a Objetos (POO) constituye un pilar esencial en el desarrollo de software moderno y en la implementación de soluciones de ciencia de datos. Este paradigma permite organizar el código en estructuras lógicas llamadas clases y objetos, promoviendo la reutilización, la modularidad y la claridad del programa. Gracias a sus principios —encapsulación, herencia y polimorfismo—, los proyectos se vuelven más escalables, colaborativos y fáciles de mantener.

Clases y Objetos Básicos

Las clases y los objetos son los componentes fundamentales de la Programación Orientada a Objetos (POO). A través de ellos, el código se organiza en estructuras que representan entidades del mundo real, facilitando la comprensión y el mantenimiento de los programas. Comprender cómo se definen y utilizan las clases, así como los atributos y métodos que las conforman, permite crear sistemas más ordenados, reutilizables y coherentes dentro de los proyectos de ciencia de datos.



Programación orientada a objetos (POO)



Clase

Es un molde o plantilla que define los atributos (datos) y métodos (funciones) que compartirán todos los objetos creados a partir de ella.



Programación orientada a objetos (POO)

Objeto

Es una instancia de una clase, con valores concretos en sus atributos y comportamientos definidos.



Atributos

Son variables que describen el estado o las características del objeto.



Métodos

Son las funciones que determinan las acciones o comportamientos que el objeto puede realizar.

Ejemplo en Python

```
# Definición de clase
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

    def mostrar(self):
        return f"Producto: {self.nombre}, Precio: {self.precio}"

# Crear objetos
p1 = Producto("Laptop", 3500)
p2 = Producto("Mouse", 50)

print(p1.mostrar())
print(p2.mostrar())
```

Las clases y los objetos constituyen la base estructural de la Programación Orientada a Objetos. Comprender su funcionamiento permite organizar el código de forma más clara y coherente, representando entidades del mundo real dentro de un programa. Su aplicación en la ciencia de datos facilita el manejo de información compleja y el diseño de modelos que reflejan la realidad de los datos de manera precisa y escalable.

CONTINUAR

Encapsulación

La encapsulación es el principio de la Programación Orientada a Objetos que consiste en ocultar los detalles internos de un objeto y controlar el acceso a sus datos mediante métodos. Este mecanismo protege la información, evita modificaciones indebidas y garantiza que las operaciones sobre los atributos se realicen de manera controlada y segura.

**VARIABLES
PRIVADAS**



Se declaran con un guión bajo _ o doble guion bajo __,

**MÉTODOS GETTER
Y SETTER**

Permiten obtener o modificar el valor de los

lo que restringe su acceso directo desde fuera de la clase.

atributos privados de forma controlada.

En el contexto de la ciencia de datos, la encapsulación resulta clave para mantener la integridad de los modelos y los datos, evitando que parámetros sensibles —como coeficientes o pesos de un algoritmo— sean alterados sin control, garantizando así resultados confiables y reproducibles.

Ejemplo en Python

```
class Cliente:
    def __init__(self, nombre, saldo):
        self.__nombre = nombre    # atributo privado
        self.__saldo = saldo

    # getter
    def get_saldo(self):
        return self.__saldo

    # setter
    def depositar(self, monto):
        if monto > 0:
            self.__saldo += monto

cliente = Cliente("Ana", 1000)
print(cliente.get_saldo())  # 1000
cliente.depositar(500)
print(cliente.get_saldo())  # 1500
```

Aplicación en ciencia de datos

Garantizar que los datos de un modelo (ej. pesos de regresión) no sean modificados directamente, sino a través de métodos controlados.

La encapsulación garantiza la seguridad e integridad de los datos, al restringir el acceso directo a los atributos de un objeto. Este principio no solo mejora la protección de la información, sino que también promueve la confiabilidad del sistema al controlar cómo se modifican los valores internos. En la ciencia de datos, esta práctica contribuye a mantener la consistencia de los modelos y la precisión de los resultados.

CONTINUAR

Herencia

La herencia es un principio fundamental de la Programación Orientada a Objetos (POO) que permite que una clase (subclase) reutilice los atributos y métodos definidos en otra clase (superclase). Este mecanismo favorece la creación de estructuras jerárquicas, donde las clases hijas heredan comportamientos comunes y pueden, además, agregar o modificar funcionalidades específicas según sus necesidades.





mantenimiento del programa.

• • •



las clases más generales sirven como base para clases más especializadas.

• • •

En el ámbito de la ciencia de datos, la herencia facilita la creación de modelos derivados de una clase base, como por ejemplo una clase *ModeloBase* que define métodos comunes de entrenamiento y evaluación, y subclases como *RegresionLineal* o *ArbolDecision* que adaptan dichos métodos a sus propios algoritmos. Este enfoque mejora la eficiencia, la claridad y la escalabilidad del código.

Ejemplo en Python

```
# Clase base
class Empleado:
    def __init__(self, nombre, salario):
        self.nombre = nombre
        self.salario = salario

    def info(self):
        return f"{self.nombre}, Salario: {self.salario}"

# Subclase
class Gerente(Empleado):
    def __init__(self, nombre, salario, departamento):
```



```
super().__init__(nombre, salario)
self.departamento = departamento

def info(self):
    return f"{self.nombre}, Salario: {self.salario}, Dept: {self.departamento}"

g1 = Gerente("Luis", 8000, "Ventas")
print(g1.info())
```

Aplicación en ciencia de datos

Implementar modelos jerárquicos: por ejemplo, `ModeloBase` (superclase) y `RegresionLineal`, `ArbolDecision` como subclases que heredan métodos de entrenamiento y evaluación.

La herencia permite construir jerarquías de clases que favorecen la reutilización y expansión del código sin necesidad de reescribirlo. Gracias a este principio, es posible crear estructuras más complejas a partir de componentes ya definidos. En proyectos de ciencia de datos, la herencia facilita la creación de modelos derivados de una clase base, optimizando el desarrollo de algoritmos y métodos de análisis.

CONTINUAR

Polimorfismo

El **polimorfismo** es un principio clave de la Programación Orientada a Objetos (POO) que permite que un mismo método adopte distintos comportamientos según el objeto que lo invoque. Gracias a esta característica, diferentes clases pueden compartir una misma interfaz, pero implementar su propia versión del método, adaptada a sus necesidades específicas.



Puede lograrse mediante sobreescritura de métodos (overriding):

Una subclase redefine un método heredado para modificar o ampliar su funcionalidad.



Favorece la flexibilidad y extensibilidad del código:

Permite incorporar nuevos tipos de objetos sin alterar la estructura general del programa.

En la ciencia de datos, el polimorfismo resulta especialmente útil al diseñar sistemas donde diversos modelos —como regresión lineal, árboles de decisión o redes neuronales— pueden compartir el mismo método de entrenamiento, aunque cada uno lo ejecute de manera distinta. Esto facilita la comparación, automatización y mantenimiento de múltiples algoritmos dentro de un mismo entorno de análisis.

Ejemplo en Python

```
class Modelo:  
    def entrenar(self):  
        pass  
  
class RegresionLineal(Modelo):  
    def entrenar(self):  
        return "Entrenando modelo de regresión lineal"  
  
class ArbolDecision(Modelo):  
    def entrenar(self):  
        return "Entrenando modelo de árbol de decisión"  
  
# Polimorfismo en acción  
modelos = [RegresionLineal(), ArbolDecision()]  
for m in modelos:  
    print(m.entrenar())
```



Aplicación en ciencia de datos

- Implementar un framework donde todos los modelos (Regresión, Árbol, Red Neuronal) usen el mismo método `entrenar()`, pero con lógicas distintas.
- Permite escribir código genérico para entrenar múltiples modelos sin cambiar la interfaz.

El polimorfismo otorga flexibilidad al código, permitiendo que un mismo método se comporte de manera diferente según el objeto que lo

implemente. Este principio simplifica la programación y favorece la adaptación de nuevas funcionalidades sin alterar la estructura general del sistema. En la práctica de la ciencia de datos, el polimorfismo permite entrenar y evaluar distintos modelos bajo una misma interfaz, agilizando el flujo de trabajo y mejorando la eficiencia del análisis.

CONTINUAR

Cuadro comparativo de los principios de la Programación Orientada a Objetos



La Programación Orientada a Objetos se fundamenta en cuatro principios esenciales que permiten estructurar el código de manera más clara, eficiente y escalable. Cada uno de estos principios —clases y objetos, encapsulación, herencia y polimorfismo— cumple una función específica que contribuye al diseño modular y a la reutilización del código.

El siguiente cuadro presenta una comparación general de estos principios, mostrando su definición, un ejemplo práctico en Python y su aplicación en el ámbito de la ciencia de datos, donde resultan claves para optimizar el desarrollo de modelos, algoritmos y sistemas inteligentes.

Principio	Definición	Ejemplo en Python	Aplicación en Ciencia de Datos
Clases y Objetos	Molde (clase) e instancia (objeto) que modelan entidades con atributos y métodos.	class Producto: ...	Modelar un registro como objeto Cliente.
Encapsulación	Oculta detalles internos y controla el acceso mediante getters/setters.	self.__saldo, get_saldo()	Proteger parámetros sensibles de modelos ML.
Herencia	Subclases reutilizan atributos y	class Gerente(Empleado): ...	Definir ModeloBase y subclases como

	métodos de superclases.		RegresiónLineal, ArbolDecision.
Polimorfismo	Un mismo método se comporta distinto según el objeto.	modelo.entrenar()	Entrenar diferentes algoritmos con la misma interfaz.

CONTINUAR

La Programación Orientada a Objetos representa un enfoque esencial para el desarrollo estructurado, modular y escalable del software en la ciencia de datos. A través de sus principios —clases y objetos, encapsulación, herencia y polimorfismo—, aprendes a organizar el código de manera lógica y reutilizable, facilitando el mantenimiento y la colaboración en proyectos complejos.



CONTINUAR

Para finalizar el tema, recordemos algunas ideas claves.

1

Clases y Objetos: son la base de la POO, permiten modelar entidades del mundo real en estructuras de datos y métodos.

2

Encapsulación: asegura integridad de datos al restringir acceso directo y permitir control a través de métodos.

3

Herencia: posibilita reutilización y extensión de código, fomentando jerarquías de clases.

4

Polimorfismo: otorga flexibilidad al permitir que un mismo método se ejecute de forma diferente según el objeto.

CONTINUAR

CIERRE

