PROGRAMACIÓN II

1º AÑO

Clase Nº 4: Programación Orientada a Objetos en Python

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Qué es la Programación Orientada a Objetos (POO)
- Diferenciar entre clases, objetos, atributos y métodos.
- Aplicar los principios de encapsulamiento, herencia y polimorfismo.
- Implementar clases simples en Python y crear instancias.
- Valorar la importancia de la POO en el desarrollo de software escalable.

1. Presentación:

Bienvenidos a la clase N.º 4 de Programación 2. Hoy comenzamos con un nuevo enfoque muy importante para el desarrollo de software: la Programación Orientada a Objetos (POO), utilizando el lenguaje Python.

Hasta ahora, hemos trabajado con estructuras básicas de programación, como funciones, listas y condicionales, resolviendo problemas en forma secuencial. Sin embargo, a medida que los programas se vuelven más grandes o que comenzamos a trabajar con estructuras más complejas —como

archivos, bases de datos, o colecciones de datos más elaboradas— necesitamos una forma de organizar mejor el código. Ahí es donde entra en juego este nuevo paradigma.

Te invito a visitar el siguiente link con un pequeño video para introducirnos en el tema.

https://www.youtube.com/watch?v=71S1WywB4cY&list=PLg9145ptuAigw5pV_DRznXdOsX19dorDs&index=2

Actividad 1: Mientras ves el video, anotá qué diferencias encontrás entre una función y un método.

La POO nos permite pensar el software como un conjunto de objetos que representan entidades del mundo real, como una persona, un producto, un sensor o incluso una colección de datos. Cada uno de estos objetos puede tener atributos (sus datos) y métodos (acciones que puede realizar). Este modelo se usa ampliamente no solo en el desarrollo de software general, sino también en el análisis de datos, donde muchas bibliotecas organizan sus herramientas utilizando clases y objetos.

Durante esta clase vamos a:

- Entender qué es una clase y cómo se crean objetos a partir de ella.
- Conocer los elementos básicos de la POO: atributos, métodos, encapsulamiento, herencia y polimorfismo.
- Escribir nuestros primeros programas orientados a objetos en Python.

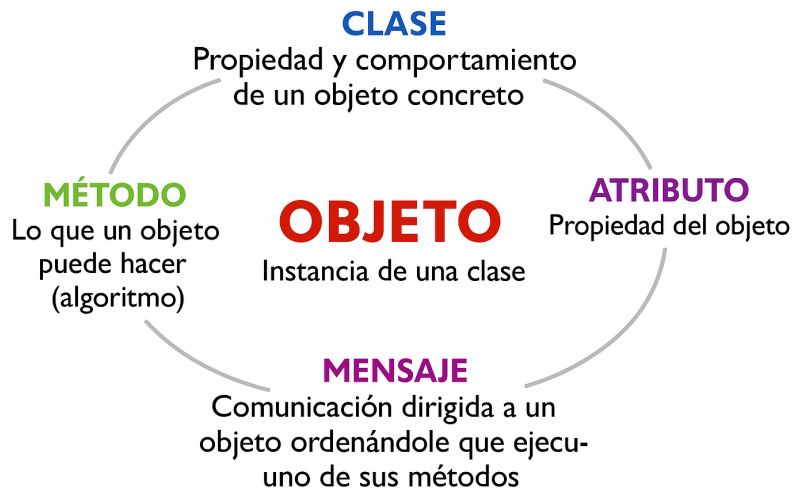
- Ver ejemplos prácticos aplicables al mundo de los datos y su organización.

Este contenido no solo es importante para mejorar tus habilidades de programación, sino que además sienta las bases para entender mejor cómo funcionan muchas de las herramientas que vas a utilizar más adelante en tu carrera. ¡Vamos a trabajar paso a paso para que este nuevo enfoque sea claro y útil desde el inicio!

2. Desarrollo

¿Qué es la Programación Orientada a Objetos?

La **Programación Orientada a Objetos (POO)** es un paradigma que surge como evolución de la programación estructurada. Está basado en el concepto de **objetos**, que combinan estructura de datos (atributos) con comportamiento (métodos). Esta combinación permite modelar sistemas de manera más natural y modular, imitando la manera en que las personas comprenden y se relacionan con el mundo real.



La POO considera que **todo en un programa puede representarse como objetos que se comunican entre sí** a través del envío de mensajes (invocación de métodos). Esto da lugar a una forma de diseñar sistemas donde cada componente es independiente, responsable de sus propias acciones y capaz de interactuar con otros objetos sin necesidad de conocer su implementación interna.

A diferencia del enfoque funcional o estructurado, donde el foco está en la secuencia de instrucciones, en la POO el foco está en **la entidad que realiza las acciones**. Esta forma de pensar favorece la reutilización de componentes, el diseño conceptual claro, la facilidad para realizar mantenimiento y la escalabilidad del sistema.

Elementos de la Programación Orientada a Objetos

Clases

Una clase es una abstracción que describe un conjunto de objetos con características y comportamientos comunes. Define un nuevo tipo de dato que agrupa tanto atributos como métodos. Actúa como una especie de "molde" o "plantilla", pero por sí sola no ocupa memoria ni tiene utilidad funcional directa. Solo al instanciar la clase se crean objetos concretos que operan en memoria.

Las clases permiten organizar el código de manera lógica, agrupando toda la funcionalidad relacionada con una entidad específica del sistema. En este sentido, son esenciales para lograr un diseño coherente y mantenible.

En Python, la definición de clases se realiza con la palabra clave *class*, seguida del nombre de la clase con mayúscula inicial, y un bloque de código indentado donde se definen los métodos y atributos.

📌 Ejemplo: definición de una clase básica en Python

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # atributo de instancia
        self.edad = edad      # atributo de instancia

    def saludar(self):
        print(f"Hola, soy {self.nombre} y tengo {self.edad} años.")
```

Una clase define la estructura y el comportamiento que compartirán todos los

objetos que se creen a partir de ella.

Objetos

Un objeto es una instancia de una clase. Cada objeto posee un conjunto de valores propios para sus atributos, lo cual determina su estado. Todos los objetos de una misma clase comparten la misma estructura, pero tienen datos distintos y existen de forma independiente en la memoria.

Por ejemplo, si definimos una clase Auto, podemos crear varios objetos: uno puede ser un Ford rojo, otro un Toyota azul, etc. Cada uno es un objeto con su propio estado, aunque comparten los mismos métodos definidos en la clase Auto.

Los objetos representan las unidades activas del programa, ya que son los que interactúan entre sí y hacen avanzar la ejecución del software.

📌 Ejemplo: creación de objetos a partir de la clase

```python
persona1 = Persona("Ana", 30)
persona2 = Persona("Luis", 25)

persona1.saludar()  # Hola, soy Ana y tengo 30 años.
persona2.saludar()  # Hola, soy Luis y tengo 25 años.
```

Un objeto es una instancia de una clase. Cada objeto tiene su propio estado, definido por sus atributos, y puede ejecutar los métodos definidos en la clase.

Atributos

Los atributos son las variables que almacenan el estado del objeto.

Pueden ser:

- De instancia: pertenecen a cada objeto individual (los más comunes).
- De clase: compartidos por todos los objetos de la misma clase.

En Python, los atributos de instancia se definen generalmente dentro del constructor *__init__*, utilizando el prefijo *self* para referirse al propio objeto.

La correcta definición y uso de atributos permite modelar con precisión las

propiedades relevantes del objeto en cuestión.

📌 Ejemplo: acceder y modificar atributos

```
print(persona1.nombre)  # Ana
persona1.edad = 31
print(persona1.edad)    # 31
```

Los atributos son variables asociadas a cada objeto. Definen su estado interno y pueden ser modificados a lo largo del tiempo.

Métodos

Los métodos son funciones definidas dentro de una clase que describen el comportamiento del objeto. Operan sobre sus atributos y pueden interactuar con otros objetos. Cada método recibe como primer parámetro la referencia al propio objeto (*self*), lo que permite acceder y modificar su estado interno.

Además de los métodos de instancia, Python permite definir:

- Métodos de clase (@classmethod): usan el parámetro cls y pueden modificar atributos de clase.

- Métodos estáticos (@staticmethod): no reciben self ni cls y actúan como funciones auxiliares.

Los métodos permiten que los objetos "respondan a mensajes" y colaboren entre sí, cumpliendo con sus responsabilidades dentro del sistema.

📌 Ejemplo: agregar un método para modificar atributos

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def cumplir_anios(self):
        self.edad += 1
        print(f"¡Feliz cumpleaños, {self.nombre}! Ahora tienes {self.edad} años.")

persona1.cumplir_anios()  # ¡Feliz cumpleaños, Ana! Ahora tienes 32 años.
```

Los métodos son funciones dentro de la clase. Permiten que los objetos realicen acciones o cambien su estado interno.

Principios fundamentales del paradigma

Encapsulamiento

El encapsulamiento es un principio central que establece que un objeto debe ocultar sus detalles internos y exponer sólo lo necesario a través de una interfaz pública. Esta ocultación se logra controlando el acceso a sus atributos mediante métodos conocidos como accesores (getters) y modificadores (setters).

El objetivo del encapsulamiento no es sólo proteger los datos, sino garantizar la integridad y coherencia del estado del objeto. Permite evitar efectos colaterales no deseados, facilita el mantenimiento y reduce el acoplamiento entre componentes.

En Python, aunque no existen modificadores de acceso estrictos como en otros lenguajes, se utilizan convenciones:

- nombre: atributo público.
- _nombre: atributo protegido.
- __nombre: atributo privado (name mangling).

Este principio también contribuye a la robustez del sistema, ya que los cambios en la implementación interna de un objeto no afectan a quienes lo utilizan, siempre que se mantenga la interfaz.

📌 Ejemplo: encapsulamiento con getters y setters

```python
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.__titular = titular    # atributo privado
        self.__saldo = saldo        # atributo privado

    def get_saldo(self):
        return self.__saldo

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

    def extraer(self, monto):
        if 0 < monto <= self.__saldo:
            self.__saldo -= monto

cuenta = CuentaBancaria("Sofía", 1000)
cuenta.depositar(500)
print(cuenta.get_saldo())  # 1500
```

El encapsulamiento oculta los detalles internos del objeto, exponiendo solo lo necesario a través de su interfaz. En Python se utiliza la convención de atributos protegidos (_atributo) y privados (__atributo).

Herencia

La herencia es el mecanismo que permite que una clase derive de otra, heredando sus atributos y métodos. La clase derivada (subclase o clase hija) puede reutilizar el comportamiento de la clase base (superclase o clase padre), agregar nuevas funcionalidades o modificar el comportamiento existente.

Este principio favorece la reutilización de código y permite construir jerarquías de clases, donde las clases más generales se especializan progresivamente en otras más concretas.

Existen distintos tipos de herencia:

- *Simple: una clase hija hereda de una sola clase padre.*
- *Múltiple: una clase hija hereda de varias clases padre (permitido en Python).*
- *Jerárquica: múltiples clases hijas heredan de una misma clase padre.*
- *Multinivel: una clase hija hereda de otra que también es hija de otra.*

Python maneja la herencia múltiple con el algoritmo MRO (Method Resolution Order), que resuelve la jerarquía de forma consistente.

La herencia permite modelar relaciones del tipo "es un": un Perro es un Animal, una Docente es una Persona.

📌 Ejemplo: herencia en acción

```python
class Empleado(Persona):  # Empleado hereda de Persona
    def __init__(self, nombre, edad, salario):
        super().__init__(nombre, edad)  # Llamamos al constructor de la superclase
        self.salario = salario

    def mostrar_info(self):
        print(f"{self.nombre}, {self.edad} años, gana ${self.salario}")

emp1 = Empleado("Carlos", 40, 120000)
emp1.saludar()        # Heredado de Persona
emp1.mostrar_info()   # Propio de Empleado
```

La herencia permite crear nuevas clases basadas en otras, reutilizando su comportamiento. En Python, se indica entre paréntesis la clase de la que se hereda.

Actividad 2: *Objetivo: Analizar un ejemplo de herencia en Python*

Instrucción: Ingresá a ChatGPT y pedile: "Dame un ejemplo de herencia en Python aplicado a una biblioteca digital". Copia el código que te devuelva y analiza:

¿Qué clase es la principal (padre)?

¿Qué clase hereda? ¿Qué nuevos atributos o métodos tiene?

Polimorfismo

El polimorfismo es la capacidad de los objetos de distintas clases de responder al mismo mensaje (método) de manera específica. Esto permite diseñar interfaces más genéricas y flexibles, ya que el mismo código puede aplicarse a múltiples tipos de objetos sin saber exactamente a cuál pertenecen.

El polimorfismo permite dos formas de implementación en Python:

- Sobreescritura de métodos (override): una clase hija redefine un método heredado para cambiar su comportamiento.

- Duck Typing: en lugar de depender del tipo del objeto, se verifica si posee los métodos esperados ("si camina como pato y hace cuac, es un pato").

Esto posibilita que distintas clases compartan un mismo contrato de uso, incluso sin herencia directa, lo que se alinea con el principio de sustitución de Liskov: una instancia de una subclase puede ser usada donde se espera una instancia de la superclase, sin alterar la lógica del programa.

📌 Ejemplo: sobreescritura de métodos

```python
class Animal:
    def hablar(self):
        print("El animal hace un sonido.")

class Perro(Animal):
    def hablar(self):
        print("Guau!")

class Gato(Animal):
    def hablar(self):
        print("Miau!")

def hacer_hablar(animal):
    animal.hablar()

hacer_hablar(Perro())  # Guau!
hacer_hablar(Gato())   # Miau!
```

El polimorfismo permite que diferentes clases implementen el mismo método de distintas maneras, lo cual permite usar los objetos de manera intercambiable.

Diseño orientado a objetos

Uno de los aspectos más poderosos de la POO es que favorece un diseño basado en abstracciones reales, lo que permite crear modelos que reflejan el dominio del problema. Este enfoque mejora la comunicación entre el equipo de desarrollo y los usuarios o clientes, y facilita el mantenimiento futuro.

El diseño orientado a objetos se basa en identificar:

- Entidades relevantes del dominio → clases.

- Sus atributos → estado.
- Sus métodos → comportamiento.
- Relaciones entre clases → herencia, asociación, composición.

Para lograr un diseño eficaz, se aplican principios SOLID, patrones de diseño y prácticas como la responsabilidad única, el bajo acoplamiento y la alta cohesión.

Cuadro resumen – Programación Orientada a Objetos en Python

Concepto	Definición	Ejemplo en Python
Clase	Plano o molde para crear objetos. Define atributos y métodos comunes.	class Persona:
Objeto	Instancia de una clase. Tiene su propio estado y comportamiento.	persona1 = Persona("Ana", 30)
Atributo	Dato que representa el estado de un objeto.	self.nombre, self.edad
Método	Función definida dentro de una clase que opera sobre sus atributos.	def saludar(self):
Encapsulamiento	Oculta detalles internos del objeto, exponiendo solo lo necesario.	__saldo, get_saldo()
Herencia	Permite que una clase herede de otra, reutilizando su comportamiento.	class Empleado(Persona):
Polimorfismo	Capacidad de usar el mismo método con comportamientos distintos.	animal.hablar() según si es Perro o Gato

Concepto	Definición	Ejemplo en Python
Instanciar	Crear un objeto a partir de una clase.	persona = Persona("Juan", 20)
Constructor	Método especial (__init__) que se ejecuta al crear el objeto.	def __init__(self, nombre):
Self	Referencia al objeto actual. Necesaria para acceder a atributos/métodos.	self.nombre = nombre

Ventajas de la Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es un paradigma que permite organizar el código en torno a entidades llamadas objetos, que combinan datos (atributos) y comportamientos (métodos). A continuación, se detallan sus principales ventajas:

- Modularidad: El código se organiza en clases, lo que permite dividir una aplicación compleja en partes más pequeñas, claras y manejables. Cada clase puede desarrollarse y probarse de forma independiente.

- Reutilización de código: Gracias a la herencia y la composición, es posible crear nuevas clases a partir de otras ya existentes, reutilizando funcionalidades sin tener que reescribir el código.

- Mantenimiento y escalabilidad: Al estar bien estructurado, el código orientado a objetos facilita el mantenimiento y la incorporación de nuevas características, permitiendo escalar el sistema con mayor facilidad.

- Modelado del mundo real: La POO se adapta naturalmente a la representación de entidades del mundo real, como personas, vehículos o productos. Esto hace que el diseño del software sea más intuitivo y comprensible.

- Encapsulamiento: Cada objeto controla el acceso a sus datos internos mediante métodos públicos, lo que reduce errores y protege la integridad del sistema. Esto mejora la seguridad y consistencia del estado interno.

- Facilidad para trabajar en equipo: Al dividir el desarrollo en clases y objetos, varios programadores pueden trabajar en paralelo en diferentes partes del sistema sin interferirse.

Actividad 3: *Modelar una clase simple*

Objetivo: Aplicar los conceptos básicos de clases, atributos y métodos.

Instrucciones:

Crear una clase llamada *Producto* que tenga los siguientes atributos:

nombre

precio

stock

Implementar los siguientes métodos:

mostrar_info(): muestra por pantalla la información del producto.

actualizar_stock(cantidad): suma o resta al stock según el valor recibido.

Opcional: Crear una instancia de la clase y probar los métodos.

INSTANCIA DE EVALUACIÓN: REGISTRO DE INTERCAMBIO: Foro Obligatorio

Casos reales de POO en YouTube

Objetivo: Explorar cómo se aplica la programación orientada a objetos en el mundo real, a través de videos educativos o de proyectos reales.

Consigna: Busca un video en YouTube donde se explique o se muestre un ejemplo de programación orientada a objetos aplicada en un proyecto real. Puede ser:

- Una aplicación hecha en Python con clases y objetos.
- Un framework que use POO (como Flask, Django, etc.).
- Un juego o simulación que modele objetos del mundo real.

Publica en el foro:

1. El link del video.
2. Un resumen máximo de 50 palabras (¿qué muestra o enseña?).
3. ¿Qué elementos de la POO se usan? ¿Qué ventajas notaste?
4. Al menos una ventaja de usar POO que se vea en el ejemplo.
5. Respondé a un compañero/a con una opinión, comparación o pregunta sobre el video que compartió.

Criterios de Evaluación

1. **Claridad y Coherencia en las Respuestas :**
    - Las respuestas deben ser claras, bien estructuradas y fáciles de seguir. Se evaluará la capacidad del estudiante para expresar sus ideas de manera coherente y lógica.
2. **Profundidad del Análisis:**
    - Se valorará la profundidad del análisis crítico en relación con el artículo. Los estudiantes deben demostrar que han comprendido los puntos clave y que pueden relacionarlos con sus propias experiencias y conocimientos.
3. **Argumentación y Justificación:**
    - Los estudiantes deben respaldar sus opiniones con argumentos sólidos y ejemplos concretos. Se evaluará la capacidad para justificar sus afirmaciones y reflexiones de manera efectiva.
4. **Participación y Colaboración en el Foro:**
    - Se tendrá en cuenta la participación activa en el foro, incluyendo la capacidad de responder a las intervenciones de otros compañeros de manera constructiva y respetuosa.

Cierre:

¡Gran trabajo en esta clase! Hoy dimos un paso clave en nuestra formación como programadores al introducirnos en la Programación Orientada a Objetos

(POO) con Python. Este paradigma no solo mejora la organización y escalabilidad del código, sino que también nos prepara para trabajar con estructuras complejas que encontraremos tanto en proyectos de análisis de datos como en el desarrollo de aplicaciones con inteligencia artificial.

Entender cómo funcionan las clases, objetos, atributos y métodos es fundamental para interpretar y utilizar muchas de las herramientas que vamos a aplicar en nuestra carrera, incluso aunque todavía no hayamos comenzado a programar modelos de IA.

En la próxima clase vamos a cambiar un poco el enfoque y comenzar a trabajar con HTML, CSS y JavaScript. Estos lenguajes forman la base del desarrollo web y son esenciales para crear interfaces visuales. Con estos conocimientos vamos a estar listos para empezar a construir nuestras propias aplicaciones web usando Flask, un framework de Python que nos permitirá integrar todo lo aprendido hasta ahora.

¡Nos espera una clase súper interesante! Hasta entonces, les deseo una excelente semana. ¡Nos vemos pronto con mucha energía y listos para seguir programando!

Bibliografía Obligatoria:

- Ramírez Jiménez, Ó. (2021). *Python a fondo*. Marcombo.

Recursos adicionales:

- Contenido: Explicaciones sobre POO, tipos de métodos, herencia, encapsulamiento, polimorfismo, abstracción, y más.

  https://ellibrodepython.com/programacion-orientada-a-objetos

- Contenido: Introducción práctica a POO, ejemplos de implementación de clases, objetos y principios fundamentales como herencia y encapsulamiento.
  https://www.datacamp.com/es/tutorial/python-oop-tutorial