

AI Engineering

Clase 2 - OOP and String Management



OOP y Gestión de Strings en Python para Tareas de Generative AI - Teoría

Duración: 45 minutos

Contexto:

- Introducción a una sesión teórica enfocada en los fundamentos de Programación Orientada a Objetos (OOP) y la gestión de strings en Python.
- Enfatiza la importancia de estos conceptos en el desarrollo de aplicaciones de Inteligencia Artificial Generativa (GenAI).



Contexto:

Relevancia de OOP y gestión de strings: Son fundamentales para estructurar y manejar proyectos complejos de GenAI de manera eficiente.

1. Introducción a la Programación Orientada a Objetos (OOP)

¿Qué es la OOP?

Definición: La **Programación Orientada a Objetos (OOP)** es un paradigma de programación que organiza el código en "objetos", los cuales son instancias de "clases". Este enfoque facilita la modularidad, la reutilización de código y la gestión de proyectos complejos.

Contexto:

- **Modularidad y Reutilización:** Permite dividir el proyecto en componentes manejables y reutilizables, esencial para proyectos de GenAI que suelen ser complejos.
- **Gestión de Proyectos Complejos:** OOP facilita la organización y mantenimiento de grandes bases de código, mejorando la eficiencia del desarrollo.

Conceptos Fundamentales de OOP

- **Clases y Objetos:**
 - **Clase:** Plantilla que define atributos (propiedades) y métodos (funciones) que los objetos crearán.
 - **Objeto:** Instancia de una clase, representando entidades individuales con valores específicos para sus atributos.
- **Atributos y Métodos:**
 - **Atributos:** Variables que almacenan el estado de un objeto.
 - **Métodos:** Funciones que definen el comportamiento de un objeto.
- **Constructores (`__init__`):**
 - Método especial que se ejecuta al crear una nueva instancia de una clase, permitiendo inicializar los atributos del objeto.

Contexto:

- **Estructuración del Código:** Facilita la creación de componentes independientes y reutilizables.
- **Definición Clara de Comportamientos:** Los métodos encapsulan funcionalidades específicas, mejorando la claridad y mantenibilidad del código.

```

class Pet:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def hablar(self):
        return "La mascota hace un sonido."

    def descripcion(self):
        return f"{self.nombre} tiene {self.edad} años."

class Dog(Pet):
    def __init__(self, nombre, edad, raza):
        # Se llama al constructor de la clase base para inicializar nombre y edad
        super().__init__(nombre, edad)
        self.raza = raza

    def hablar(self):
        # Sobrescribimos el método hablar para los perros
        return "Guau guau!"

    def descripcion(self):
        return f"{self.nombre} es un {self.raza} de {self.edad} años."


if __name__ == "__main__":

    mascota_general = Pet("Bobby", 3)
    print(mascota_general.descripcion())
    print("Habla:", mascota_general.hablar())

    print("-" * 40)

    perro = Dog("Max", 5, "Labrador")
    print(perro.descripcion())
    print("Habla:", perro.hablar())

```



```
class ModeloGenerativo:
    def __init__(self, nombre, version):
        self.nombre = nombre
        self.version = version

    def cargar_modelo(self):
        print(f"Cargando modelo {self.nombre} versión {self.version}")

    def generar_texto(self, prompt):
        print(f"Generando texto para el prompt: {prompt}")
```

2. Principios de la OOP

- **Encapsulación**

Definición: La encapsulación consiste en ocultar la complejidad interna de los objetos, exponiendo solo lo necesario a través de interfaces públicas. Mejora la seguridad y la integridad de los datos, evitando modificaciones no deseadas desde fuera de la clase.

- **Herencia**

Definición: La herencia permite que una clase (subclase) herede atributos y métodos de otra clase (superclase), promoviendo la reutilización de código y la creación de jerarquías lógicas.

- **Polimorfismo**

Definición: El polimorfismo permite que objetos de diferentes clases sean tratados de manera uniforme a través de una interfaz común, permitiendo la flexibilidad en el comportamiento de los objetos.

3. Patrones de Diseño en OOP

Definición: Los patrones de diseño son soluciones reutilizables a problemas comunes en el desarrollo de software. Implementar estos patrones en proyectos de GenAI puede mejorar la estructura, la escalabilidad y la eficiencia del código.

- Singleton
- Factory
- Strategy

Singleton

Definición: Garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a ella. Una metaclass clase de la clase. Es decir, una clase es una instancia de la metaclass..

Importancia en GenAI:

- **Gestión Centralizada de Recursos Críticos:** Asegura que componentes como conexiones a APIs o recursos compartidos sean gestionados de manera centralizada, evitando conflictos y redundancias.

```
class SingletonMeta(type):  
    _instances = {}  
  
    def __call__(cls, *args, **kwargs):  
        if cls not in cls._instances:  
            cls._instances[cls] = super().__call__(*args, **kwargs)  
        return cls._instances[cls]  
  
class Configuracion(metaclass=SingletonMeta):  
    def __init__(self, api_key):  
        self.api_key = api_key
```

Factory

Definición: Proporciona una interfaz para crear objetos en una superclase, pero permite que las subclasses alteren el tipo de objetos que se crearán.

Importancia en GenAI:

- **Creación Flexible de Modelos:** Facilita la creación de diferentes tipos de modelos o herramientas sin necesidad de conocer las clases específicas en tiempo de ejecución, permitiendo una mayor modularidad y adaptabilidad.

```
class ModelFactory:

    @staticmethod
    def crear_modelo(tipo, nombre, version, **kwargs):
        if tipo == "GPT":
            return ModeloGPT(nombre, version, kwargs.get("api_key"))
        elif tipo == "OtroModelo":
            return OtroModelo(nombre, version, kwargs.get("parametro"))
        else:
            raise ValueError(f"Tipo de modelo '{tipo}' no reconocido.")
```

Strategy

Definición: Define una familia de algoritmos, encapsula cada uno y los hace intercambiables. Permite que el algoritmo varíe independientemente de los clientes que lo utilizan.

Importancia en GenAI:

- **Cambio Dinámico de Estrategias:** Permite modificar las estrategias de generación de texto o procesamiento de datos según las necesidades específicas del proyecto, mejorando la flexibilidad y adaptabilidad del sistema.

```
class EstrategiaGeneracion:
    def generar(self, prompt):
        raise NotImplementedError

class EstrategiaSimple(EstrategiaGeneracion):
    def generar(self, prompt):
        return f"Generación simple para: {prompt}"

class EstrategiaAvanzada(EstrategiaGeneracion):
    def generar(self, prompt):
        return f"Generación avanzada para: {prompt}"
```

4. Gestión de Strings en Python

Manipulación de Strings

Definición: Las strings son fundamentales en GenAI, ya que gran parte de las tareas involucran la generación, procesamiento y transformación de texto. Python ofrece una variedad de métodos para manipular strings de manera eficiente.

Técnicas Clave:

- **Métodos de Strings:** `.strip()`, `.capitalize()`, `.lower()`, `.upper()`, `.replace()`, `.split()`, `.join()`, entre otros.
- **Formateo de Strings:** Utilización de f-strings, `.format()`, y plantillas para construir prompts dinámicos.

```
texto = "  Hola Mundo  "
texto_limpio = texto.strip().capitalize()
print(texto_limpio)  # Output: "Hola mundo"
```

Formateo y Construcción de Prompts

Definición: La calidad de los prompts afecta directamente la efectividad de los modelos de lenguaje. Crear prompts claros, coherentes y bien estructurados es esencial para obtener resultados óptimos.

Importancia en GenAI:

- **Guía Adecuada para los Modelos:** Prompts bien diseñados dirigen al modelo generativo hacia respuestas más precisas y relevantes.
- **Consistencia en los Inputs:** Mejora la calidad y coherencia de las respuestas generadas.

```
tema = "inteligencia artificial en la educación"
punto_clave = "beneficios de personalizar el aprendizaje"
prompt = f"Escribe un artículo sobre {tema} que incluya {punto_clave}."
print(prompt)

# Output: "Escribe un artículo sobre inteligencia artificial en la educación que incluya
beneficios de personalizar el aprendizaje."
```

Optimización de Strings

Definición: Optimizar la manipulación de strings mejora la eficiencia del procesamiento y la generación de texto, lo cual es crucial en aplicaciones de alta demanda como chatbots, generadores de contenido y sistemas de recomendación.

Técnicas de Optimización:

- **Uso de Expresiones Regulares (regex):** Para patrones complejos.
- **Evitar Operaciones Costosas dentro de Bucles:** Mejora el rendimiento general.
- **Utilizar Métodos Nativos de Python:** Aprovecha las optimizaciones internas de Python.

```
import re

texto = "El precio es de $300 y se incrementará a $350."
precios = re.findall(r'\$\d+', texto)
print(precios)  # Output: ['$300', '$350']
```

5. Aplicación de OOP y Gestión de Strings en GenAI

Modelado de Componentes de GenAI con OOP

Definición: Utilizar clases para representar modelos, datasets, pipelines y otras entidades permite una estructura clara y modular del proyecto, facilitando su desarrollo y mantenimiento.

```
class Dataset:
    def __init__(self, nombre, datos):
        self.nombre = nombre
        self.datos = datos

    def cargar_datos(self):
        print(f"Cargando datos para el dataset: {self.nombre}")

class Pipeline:
    def __init__(self, nombre, dataset):
        self.nombre = nombre
        self.dataset = dataset

    def ejecutar(self):
        self.dataset.cargar_datos()
        print(f"Ejecutando pipeline: {self.nombre}")
```


Gestión Eficiente de Prompts

Definición: Implementar métodos de limpieza, formateo y construcción de prompts asegura que las interacciones con los modelos de lenguaje sean efectivas y coherentes, mejorando la calidad de los resultados generados.

```
class ModeloGenerativo:
    def __init__(self, nombre, version):
        self.nombre = nombre
        self.version = version

    def limpiar_prompt(self, prompt):
        prompt = prompt.strip().capitalize()
        return prompt

    def generar_texto(self, prompt):
        prompt_limpio = self.limpiar_prompt(prompt)
        return f"Generando texto para: {prompt_limpio}"
```

Integración con Librerías de GenAI

Definición: La combinación de OOP y gestión de strings facilita la integración con librerías y frameworks populares como TensorFlow, PyTorch, Langchain, entre otros, permitiendo aprovechar al máximo sus capacidades en proyectos de GenAI.

```
from langchain import LLM

class ModeloLangchain(ModeloGenerativo):
    def __init__(self, nombre, version, api_key):
        super().__init__(nombre, version)
        self.api_key = api_key
        self.llm = LLM(api_key=self.api_key)

    def generar_texto(self, prompt):
        prompt_limpio = self.limpiar_prompt(prompt)
        respuesta = self.llm.generate(prompt_limpio)
        return respuesta
```

Importancia de los Conceptos en GenAI

OOP y Estructura del Código

- **Sistemas Modulares:**
 - Permite que cada componente pueda ser desarrollado, probado y mantenido de manera independiente.
 - Facilita la colaboración en equipo, ya que diferentes desarrolladores pueden trabajar en distintas clases o módulos sin interferir entre sí.
- **Facilidad de Colaboración:**
 - Mejora la coordinación entre desarrolladores al proporcionar una estructura clara y definida para el código.
 - Reduce conflictos y facilita la integración de diferentes partes del proyecto.

Gestión de Strings y Prompts

- **Interacción Mejorada con Modelos de Lenguaje:**
 - Asegura que los prompts sean claros y estructurados, mejorando la relevancia y precisión de las respuestas.
- **Optimización del Rendimiento:**
 - Reduce tiempos de respuesta y mejora la eficiencia del sistema al manejar textos de manera óptima.

Patrones de Diseño y Escalabilidad

- **Expansión Facilitada:**
 - Permite añadir nuevas funcionalidades sin comprometer la estabilidad del sistema.
- **Mantenibilidad Mejorada:**
 - Reduce la complejidad y facilita la identificación y corrección de errores, mejorando la calidad del código.

PREGUNTAS

