

CARRERA DE COMPUTACIÓN

INFORME

1. Datos informativos

1.1.Módulo: 1

1.2.Nivel: Séptimo

1.3.Fecha: 10 de marzo de 2025

1.4.Nombres: Aupas Antony, Baraja Cristian, Basantes Geovanny

1.5.Tema: Paradigmas de la Programación

2. Objetivo

Comprender los principales paradigmas de programación, sus características, ventajas y desventajas, y aplicar conceptos básicos mediante ejercicios prácticos.

3. Contenido

3.1.Introducción a los Paradigmas de Programación

3.1.1. Definición de paradigma de programación

Según Martínez (2020) los paradigmas de programación representan distintos estilos de desarrollo de software, cada uno documentado y diseñado para resolver problemas computacionales según las necesidades específicas. Estos enfoques proporcionan diversas filosofías y métodos para abordar la solución de problemas en el ámbito de la programación.

3.1.2. Importancia en el desarrollo de software

Los paradigmas definen cómo los programadores piensan sobre la solución de problemas, facilitando la organización del código y la mejora de la productividad. Por otro lado, cada paradigma Brinda diferentes ventajas según el tipo de problema a solucionar, lo que ayuda en la optimización del rendimiento, el mantenimiento y la escalabilidad de los sistemas (Sommerville, 2020).

3.1.3. Breve historia de los paradigmas de programación

Los paradigmas de programación han tenido su evolución junto con la historia de la computación. Al principio, la programación se realizaba en lenguaje máquina, llegando a tener una complejidad a nivel alto y de igual forma poca eficiencia. Con el tiempo, surgieron lenguajes de

alto nivel y nuevos enfoques, como la programación estructurada en la década de 1960, que mejoró la legibilidad y organización del código (Dijkstra, 1968). Posteriormente, en los años 1980, la programación orientada a objetos revolucionó el desarrollo de software al introducir conceptos como encapsulamiento y reutilización de código (Booch, 1994). Más recientemente, paradigmas como la programación funcional y la programación orientada a eventos han cobrado relevancia, adaptándose a las nuevas demandas de la computación distribuida y la inteligencia artificial (Van y Haridi, 2003).

3.2. Principales Paradigmas

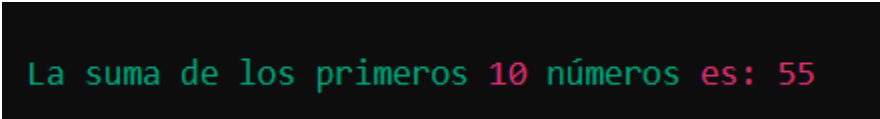
3.2.1. Programación Imperativa

En la programación imperativa, el enfoque se basa en la ejecución secuencial de instrucciones que modifican el estado del programa.

- **Ejemplo:** Lenguajes como C, Pascal.
- **Ejercicio:** Implementación de un algoritmo simple en C. Ejemplo: Crear un programa que calcule la suma de los primeros 10 números.
- **Código:**

```
#include <stdio.h>
int main() {
    int suma = 0;
    for (int i = 1; i <= 10; i++) {
        suma += i;
    }
    printf("La suma de los primeros 10 números es: %d\n", suma);
    return 0;
}
```

Salida esperada:



```
La suma de los primeros 10 números es: 55
```

3.2.2. Programación Orientada a Objetos (POO)

- **Conceptos clave:**

- **Clases:** Plantillas para crear objetos.
- **Objetos:** Instancias de clases.
- **Herencia:** Capacidad de crear nuevas clases basadas en clases existentes.
- **Polimorfismo:** Habilidad de un objeto para tomar múltiples formas.
- **Lenguajes representativos:** Java, Python, C++.
- **Ejercicio:** Creación de una clase en Python o Java. Ejemplo: Definir una clase "Coche" con atributos y métodos básicos.
- **Código:**

```
public class Coche {
    private String marca;
    private String modelo;
    private int año;

    public Coche(String marca, String modelo, int año) {
        this.marca = marca;
        this.modelo = modelo;
        this.año = año;
    }

    public String mostrarInfo() {
        return "Coche: " + marca + " " + modelo + ", Año: " + año;
    }

    public static void main(String[] args) {
        Coche miCoche = new Coche("Toyota", "Corolla", 2022);
        System.out.println(miCoche.mostrarInfo()); // Salida: Coche: Toyota Corolla, Año:
2022
    }
}
```

3.2.3. Programación Funcional

- **Principios:**
 - **Inmutabilidad:** Los datos no cambian una vez creados.
 - **Funciones de orden superior:** Las funciones pueden recibir otras funciones como parámetros o devolverlas como resultados.
 - **Recursión:** Una función se llama a sí misma para resolver un problema.
- **Lenguajes representativos:** Haskell, Lisp, Elixir.

- **Ejercicio:** Implementación de una función recursiva en Python o Haskell. Ejemplo: Crear una función recursiva que calcule el factorial de un número.

- **Código:**

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)
```

Ejemplo de uso

```
print(factorial(5))
```

Salida: 120

3.2.4. Programación Lógica

La programación lógica se basa en la formalización del conocimiento y la inferencia automática. Se utiliza para resolver problemas que requieren deducción y búsqueda.

- **Lenguaje representativo:** Prolog.
- **Ejercicio:** Consulta simple en Prolog. Ejemplo: Definir hechos y reglas sobre familiares y hacer consultas sobre la relación entre personas.

- **Código:**

% Hechos: Definimos relaciones familiares básicas

```
padre(juan, maria).
```

```
padre(juan, pedro).
```

```
madre(ana, maria).
```

```
madre(ana, pedro).
```

```
padre(carlos, juan).
```

```
madre(luisa, juan).
```

% Reglas: Definimos relaciones derivadas

```
hermano(X, Y) :- padre(P, X), padre(P, Y), madre(M, X), madre(M, Y), X \= Y.
```

```
abuelo(X, Y) :- padre(X, P), padre(P, Y).
```

```
abuelo(X, Y) :- padre(X, P), madre(P, Y).
```

% Consultas de ejemplo:

```
% ?- hermano(maria, pedro). % Respuesta: true
```

```
% ?- abuelo(carlos, maria). % Respuesta: true
```

```
% ?- abuelo(luisa, pedro). % Respuesta: true
```

3.3. Comparación de Paradigmas

3.3.1. Ventajas y desventajas de cada paradigma

Tabla 1. Ventajas y desventajas de los paradigmas de programación

Tipo de Programación	Ventaja	Desventaja
Programación Imperativa	Simple y directa	Difícil de mantener en proyectos grandes
Programación Orientada a Objetos	Modularidad y reutilización	Puede ser compleja de entender al principio
Programación Funcional	Manejo sencillo de datos inmutables y ejecución paralela	Puede ser difícil de aprender y menos eficiente en algunos casos
Programación Lógica	Adecuado para problemas de inferencia	Rendimiento limitado en problemas no lógicos

3.3.2. Casos de uso según el tipo de problema:

- Programación imperativa es ideal para tareas secuenciales y de bajo nivel.
- POO es perfecta para sistemas complejos que requieren modularidad y reutilización.
- Programación funcional es adecuada para tareas de procesamiento de datos y operaciones concurrentes.
- Programación lógica es útil en inteligencia artificial y sistemas expertos.

3.3.3. Discusión sobre tendencias actuales:

Existen paradigmas híbridos que combinan características de varios paradigmas, como el multiparadigma, que busca aprovechar lo mejor de cada enfoque.

3.4. Actividad Práctica

3.4.1. Presentación de un problema real (Relación a su proyecto de clase):

Modelo de PLN (Procesamiento de Lenguaje Natural)

En el paradigma funcional, el modelo de PLN se implementaría como una función pura que toma un texto como entrada y devuelve un resultado sin modificar ningún estado externo. Además, se pueden utilizar técnicas de explicabilidad como LIME o SHAP, que también se implementarían como funciones puras.

Implementación:

```
# Función pura para procesar texto con un modelo preentrenado (BERT, GPT, etc.)
def procesar_texto(modelo, texto):
    # Simulación de procesamiento con un modelo de IA
    resultado = modelo(texto) # Aquí se aplicaría el modelo de PLN
    return resultado

# Función pura para explicar la decisión del modelo usando LIME o SHAP
def explicar_decision(modelo, texto):
    # Simulación de explicación usando técnicas como LIME o SHAP
    explicacion = f"El modelo decidió que '{texto}' es relevante debido a X, Y, Z."
    return explicacion

# Ejemplo de uso
texto_ejemplo = "Este es un texto de ejemplo."
modelo_bert = lambda x: "Resultado positivo" # Simulación de un modelo preentrenado
resultado = procesar_texto(modelo_bert, texto_ejemplo)
explicacion = explicar_decision(modelo_bert, texto_ejemplo)

print("Resultado:", resultado)
print("Explicación:", explicacion)
```

Para el backend, se utilizaría un framework funcional como FastAPI, que permite definir endpoints como funciones puras. Cada endpoint recibirá una solicitud, procesará los datos y devolverá una respuesta sin efectos secundarios.

Implementación:

```

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

app = FastAPI()

# Modelo de datos para la solicitud (inmutable)
class TextoEntrada(BaseModel):
    texto: str

# Función pura para el endpoint de análisis de texto
@app.post("/analizar-texto")
def analizar_texto(solicitud: TextoEntrada):
    texto = solicitud.texto
    resultado = procesar_texto(modelo_bert, texto) # Usamos la función pura definida antes
    explicacion = explicar_decision(modelo_bert, texto) # Función pura de explicación
    return {"resultado": resultado, "explicacion": explicacion}

# Ejemplo de uso (simulación)
# Si ejecutas este código con FastAPI, podrías hacer una solicitud POST a /analizar-texto con el siguiente
# JSON como:
# {"texto": "Este es un texto de ejemplo."}

```

En el paradigma funcional, las operaciones de base de datos se manejarían como funciones puras que devuelven datos sin modificar el estado global. Para MongoDB y PostgreSQL, se utilizarían funciones que encapsulan las consultas.

```

# Función pura para guardar texto en MongoDB
def guardar_texto_en_mongodb(texto):
    # Simulación de guardado en MongoDB
    return {"status": "success", "mensaje": f"Texto '{texto}' guardado en MongoDB"}

# Función pura para obtener metadatos de PostgreSQL
def obtener_metadatos_de_postgresql(id):
    # Simulación de consulta a PostgreSQL
    return {"id": id, "metadatos": "Información relevante"}

# Ejemplo de uso
texto = "Este es un texto de ejemplo."
resultado_mongo = guardar_texto_en_mongodb(texto)
metadatos_postgres = obtener_metadatos_de_postgresql(1)

print(resultado_mongo)
print(metadatos_postgres)

```

3.4.2. Implementación de una solución en diferentes paradigmas:

Paradigma 1: Explicabilidad en Modelos de IA con NLP usando WordPress + PHP (Monolítico)

Backend:

- Usando **WordPress** como backend para gestionar los datos relacionados con los modelos de IA (como los resultados de los análisis de texto).
- Al crear un **Custom Post Type (CPT)** para almacenar los resultados de los análisis de los modelos de NLP, como las predicciones de clasificación de texto, la detección de emociones, etc. Además, agregas **campos personalizados** para detalles de las predicciones (por ejemplo, la probabilidad de cada clase, las características clave extraídas, etc.).

Frontend:

- Usas **PHP** dentro de WordPress para mostrar los resultados de los modelos de IA en una página.
- Los usuarios pueden ver cómo se procesaron los textos y entender las predicciones del modelo directamente desde el sitio web. El frontend se genera con **PHP** y se muestra en las páginas de WordPress.

Ventaja:

- **Fácil de implementar:** todo el sistema (gestión de datos y visualización de resultados) está dentro de **WordPress**, lo que hace que sea sencillo gestionarlo y mantenerlo.
- **Útil para prototipos:** si necesitas crear rápidamente una plataforma de demostración con explicaciones simples para los resultados de los modelos de NLP, este enfoque es eficiente.

Desventaja:

- **Menos flexibilidad:** todo está dentro de **WordPress**, lo que dificulta la personalización avanzada del diseño y la visualización de los modelos de IA.
- **Limitado en interactividad:** puede ser complicado integrar interacciones más complejas para explorar cómo los modelos de NLP toman decisiones (por ejemplo, visualizaciones interactivas de atención, palabras clave relevantes, etc.).

Paradigma 2: Explicabilidad en Modelos de IA con NLP usando WordPress + React (Desacoplado)

Backend:

- Usas **WordPress** solo para gestionar los datos de los modelos de IA, como los resultados de las predicciones de NLP.
- Expones los resultados de las predicciones y análisis de NLP a través de la **API REST de WordPress**. Esto permite que la información de los modelos de IA (por ejemplo, clasificación de texto, análisis de sentimiento) sea accesible desde cualquier frontend.

Frontend:

- Usas **React** para crear una interfaz frontend más dinámica e interactiva.
- El frontend de **React** consume la API REST de WordPress para obtener los resultados del modelo de NLP y presentarlos de una manera más interactiva.
- Los usuarios pueden explorar las explicaciones de los modelos de IA, como visualizar los detalles sobre qué palabras clave fueron relevantes para las decisiones del modelo, cómo se clasificaron los textos, etc.

Ventaja:

- **Más flexible y escalable:** puedes construir interfaces más complejas y dinámicas en React, mostrando visualizaciones interactivas sobre cómo el modelo de NLP toma decisiones (por ejemplo, resaltando las palabras que el modelo considera importantes).
- **Mejor experiencia de usuario:** el frontend desacoplado permite crear interfaces modernas, rápidas y atractivas que facilitan la comprensión de los resultados de los modelos de IA.
- **Mayor control sobre la interacción:** puedes integrar herramientas de explicabilidad como **LIME**, **SHAP** o **visualizaciones de atención** para explicar cómo los modelos de NLP llegaron a una conclusión.

Desventaja:

- **Mayor complejidad:** necesitas conocimientos tanto de WordPress (para gestionar los datos) como de React (para crear el frontend y consumir la API). La implementación es más compleja que el paradigma monolítico.
- **Requiere más infraestructura:** deberás gestionar tanto el backend (WordPress) como el frontend (React), lo cual puede ser más costoso y requerir más mantenimiento.

4. Conclusiones

En resumen, los paradigmas de programación (imperativo, orientado a objetos, funcional, lógico) tiene sus propias características, ventajas y desventajas. Dependiendo del tipo de problema, un paradigma puede ser más adecuado que otro. Por ejemplo, la programación orientada a objetos es excelente para sistemas complejos, mientras que la programación funcional es ideal para tareas que requieren inmutabilidad de datos y procesamiento paralelo.

También hay que tener en cuenta que la programación multiparadigma surge como una alternativa versátil que permite combinar distintos enfoques dentro de un mismo lenguaje, facilitando su adaptación a las necesidades específicas de cada proyecto. Lenguajes como Python, C++ y JavaScript sobresalen por su capacidad de integrar múltiples paradigmas, proporcionando a los desarrolladores herramientas más flexibles. En un entorno tecnológico en constante evolución, dominar y aplicar estos enfoques contribuye a optimizar la calidad del software, aprovechar mejor los recursos y seleccionar la estrategia más adecuada para resolver cada desafío.

Por otro lado, los paradigmas híbridos, como el uso de WordPress con React, permiten combinar lo mejor de ambos mundos. El backend en WordPress facilita la gestión de datos, mientras que el frontend desacoplado en React mejora la interactividad y personalización de la interfaz de usuario, creando una experiencia más dinámica y eficiente. Sin embargo, este enfoque puede requerir más infraestructura y conocimientos técnicos.

5. Referencias Bibliográficas

Booch. (1994). *Object-Oriented Analysis and Design with Applications*. Obtenido de <https://zjnu2017.github.io/OOAD/reading/Object.Oriented.Analysis.and.Design.with.Applications.3rd.Edition.by.Booch.pdf>

- Dijkstra, E. (1968). *Go To Statement Considered Harmful*. Obtenido de <https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf>
- Martínez, M. (9 de Junio de 2020). *Profile*. Obtenido de ¿Qué son los paradigmas de programación?: <https://profile.es/blog/que-son-los-paradigmas-de-programacion/>
- Morales, K. G. (2003). Fundamentos de programación.
- Sommerville, I. (2020). *Software engineering*. Obtenido de <https://dn790001.ca.archive.org/0/items/bme-vik-konyvek/Software%20Engineering%20-%20Ian%20Sommerville.pdf>
- Van, P., & Haridi, S. (5 de Junio de 2003). *Concepts, Techniques, and Models of Computer Programmin*. Obtenido de <https://webperso.info.ucl.ac.be/~pvr/VanRoyHaridi2003-book.pdf>