Guía de resolución de problemas Nº 1

Unidad 1

Historia de los lenguajes de programación. Introducción a paradigmas de programación: estructurada, orientada a objetos, funcional y lógica. Programación orientada a agentes.

Parte A – Preguntas

1. ¿Qué disciplina o buena práctica exige cada paradigma (estructurado, OO y funcional) al programador?
2. ¿Qué significa que un lenguaje sea multiparadigma?
3. ¿Qué es una función pura?
4. ¿Qué instrucción se considera "prohibida" o desaconsejada en la programación estructurada porque rompe el flujo secuencial del programa?
5. ¿Qué ventajas ofrece la POO para construir software de análisis y procesamiento de datos que deba escalar y mantenerse en el tiempo?

Parte B – Problemas

Problema 1

Dado los siguientes fragmentos de código en Python, identifique qué paradigma(s) de programación se están utilizando. Justifique su respuesta.

```python
# Fragmento A
numeros = [1, 2, 3, 4, 5]
cuadrados = list(map(lambda x: x**2, filter(lambda x: x % 2 == 0, numeros)))
print(cuadrados)
```

```python
# Fragmento B
class Calculadora:
    def __init__(self):
        self.resultado = 0

    def sumar(self, a, b):
        self.resultado = a + b
        return self.resultado

calc = Calculadora()
print(calc.sumar(3, 5))
```

```python
# Fragmento C
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

```
# Fragmento D
% Hechos: Verdades base del sistema
padre('Homero', 'Bart').
padre('Homero', 'Lisa').
padre('Abraham', 'Homero').

% Reglas: Relaciones que se deducen a partir de los hechos
% X es abuelo de Y si X es padre de Z, y Z es padre de Y
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).
```

Problema 2

Completar la siguiente tabla.

Característica	Funcional	Orientada a Objetos	Estructurada
¿En qué se basa?			
¿Cómo maneja los datos?			
¿Cómo se reúsa el código?			
Ejemplos de lenguajes			
Ventaja principal			
Desventaja principal			

Problema 3

Una universidad ha decidido modernizar el funcionamiento de su biblioteca mediante el desarrollo de un sistema digital que permita administrar el catálogo bibliográfico y el préstamo de material a estudiantes y docentes.

Actualmente la biblioteca realiza gran parte de sus tareas de manera manual, lo que genera dificultades para mantener actualizado el inventario de ejemplares, controlar los préstamos vigentes y conocer la disponibilidad real de los materiales.

El nuevo sistema deberá permitir registrar a las personas autorizadas a utilizar la biblioteca, almacenar información sobre los libros disponibles y administrar las operaciones de préstamo y devolución.

Cada libro posee información bibliográfica como título, ISBN, año de publicación y editorial. Además, un mismo libro puede tener varios autores asociados y existir en múltiples ejemplares físicos dentro de la biblioteca. Cada ejemplar posee un código de identificación único y un estado que indica si se encuentra disponible, prestado o en reparación.

Las personas registradas en el sistema podrán solicitar préstamos de ejemplares disponibles. Para cada préstamo se deberá almacenar la fecha en que fue realizado y la fecha prevista para su devolución. Cuando un ejemplar es devuelto, el sistema deberá actualizar automáticamente su disponibilidad.

La biblioteca también desea conocer en todo momento qué usuarios poseen préstamos activos y qué ejemplares se encuentran disponibles para ser prestados.

En una futura versión del sistema se prevé incorporar distintos tipos de usuarios, como estudiantes de grado, estudiantes de posgrado y docentes, cada uno con diferentes límites de préstamos simultáneos.

Actividades

a) Identifique los principales posibles objetos del dominio

b) Para cada objeto identificado, determine posibles atributos

c) Determine las principales responsabilidades que podría tener cada objeto

Unidad 2

Clases y objetos. Atributos, métodos y métodos especializados. Abstracción. Encapsulamiento. Modularidad. Jerarquías. Tipos. Concurrencia de objetos. Persistencia. Control de acceso. Clases abstractas. Polimorfismo. Relaciones.

Parte A – Preguntas

1. ¿Cuál es la diferencia entre clase y objeto? ¿Y entre un atributo y un método?
2. ¿Qué es una abstracción?
3. ¿Qué ventajas aporta el encapsulamiento?
4. ¿Qué diferencia existe entre composición y herencia?
5. ¿Qué significa polimorfismo?
6. ¿Cuándo conviene utilizar una clase abstracta?
7. ¿Cómo se implementa el encapsulamiento en Python si no existe el modificador private estricto? ¿Cuál es la convención y cuál el mecanismo técnico?
8. ¿Qué es la persistencia de objetos y por qué es relevante en un sistema que procesa grandes volúmenes de datos?

Parte B – Problemas

Problema 1

Definir una clase base `Vehiculo` con método `calcular_combustible()`. Crear dos subclases: `Auto` (combustible = distancia * 0.1) y `Moto` (combustible = distancia * 0.05). Escribir una función que reciba una lista de vehículos y calcule el combustible total para una distancia dada, haciendo uso de polimorfismo.

Problema 2

Diseña e implementa una clase `DatoCientifico` que represente una medición de un experimento. Debe contener:

- Atributos privados (por convención): `__valor`, `__unidad`, `__timestamp`, `__fuente`.
- Propiedades (@property) para acceso controlado.
- Métodos especializados: `__str__`, `__eq__` (dos datos son iguales si tienen mismo valor, unidad y fuente), `__lt__` (comparar por timestamp).
- Un método de clase `@classmethod` para crear una instancia desde una cadena con formato "valor;unidad;timestamp;fuente".
- Un método estático `@staticmethod` para validar que una unidad pertenezca a un catálogo permitido.

Problema 3

Diseñar una jerarquía de clases para normalizar distintos tipos de datos dentro de un dataset. La clase madre debe ser una clase abstracta `Dato` con los métodos abstractos `es_valido()` y `normalizar()`. A partir de ella, implementar tres subclases concretas: `DatoNumerico`, `DatoCategorico` y `DatoTextual`.

Cada subclase debe implementar normalizar() de forma polimórfica: en `DatoNumerico` debe escalar el rango a [0, 1] posibles valores de entrada entre 0 y 100, en `DatoCategorico` debe aplicar una codificación ordinal (definida por ustedes), y `DatoTextual` debe eliminar espacios y convertir todo a minúsculas. Asimismo, cada una debe implementar `es_valido()` con criterios coherentes para su tipo (por ejemplo, descartar datos nulos, categorías desconocidas o cadenas vacías).

Finalmente, implementar una función `normalizar_dataset(lista_datos)` que reciba una lista de objetos `Dato`, verifique que cada uno sea válido mediante `es_valido()`, invoque `normalizar()` sin reconocer el tipo de dato concreto y devuelva la lista normalizada. Incluyan un prueba con al menos dos instancias de cada subclase para mostrar el resultado antes y después de la normalización.

Problema 4

Analice las siguientes situaciones e identifique el tipo de relación (asociación, agregación, composición, herencia, dependencia):

a. Un `Auto` tiene un `Motor`. Si se destruye el auto, el motor también deja de existir.
b. Un `Profesor` imparte varias `Materias`. Si el profesor deja de existir, las materias siguen existiendo.
c. Un `Estudiante` usa una `Calculadora` para resolver ejercicios.

d. Un `Cuadrado` es un tipo especial de `Rectangulo`.
e. Un `Pedido` contiene varios `Productos`.

Implementa en Python al menos dos de estas relaciones, justificando la elección de agregación vs. composición.

Unidad 3

Introducción a UML para representaciones básicas. Análisis orientado a objetos. Diseño orientado a objetos. Diseño dirigido por responsabilidades. Diseño por contratos: Precondiciones, postcondiciones e invariantes. Principios SOLID. Patrones de diseño.

Parte A – Preguntas

1. ¿Cuál es la diferencia entre análisis y diseño orientado a objetos?
2. ¿Qué representa un diagrama de clases?
3. ¿Qué es una responsabilidad?
4. ¿Qué elementos conforman un contrato en el contexto de la programación? ¿Cómo se expresan precondiciones, postcondiciones e invariantes en código Python?
5. ¿Qué diferencia existe entre precondición, postcondición e invariante?
6. ¿Qué principio SOLID se viola si una clase Pipeline se encarga de ejecutar etapas, guardar archivos, generar reportes y conectarse a redes? ¿Cómo lo resolverías?
7. ¿Qué ventajas ofrecen los patrones de diseño?
8. Explique e implemente con un ejemplo sencillo, tres patrones de diseño.

Parte B – Problemas

Problema 1

Una empresa de análisis de datos recibe diariamente archivos en formato CSV con información proveniente de distintos laboratorios. Para procesar esos archivos, un desarrollador implementó la siguiente clase:

```
class Pipeline:
    def procesar(self, archivo):
        # Leer archivo CSV
        ...
        # Validar los registros
        ...
        # Eliminar registros inválidos
        ...
        # Exportar el resultado a JSON
```

```
…
```

Con el paso del tiempo comenzaron a surgir nuevos requerimientos:

- Los archivos ya no siempre serán CSV; también podrán recibirse en otros formatos.
- Las reglas de validación cambian con frecuencia.
- Además de JSON, será necesario exportar los resultados en formato XML.

El equipo de desarrollo considera que la clase Pipeline será cada vez más difícil de mantener y modificar.

Actividades

1. Analice la clase presentada e indique qué principio de diseño orientado a objetos se está incumpliendo. Justifique su respuesta.
2. Proponga un nuevo diseño que distribuya las responsabilidades entre distintas clases.
3. Para cada clase propuesta, indique:
    a. su responsabilidad;
    b. los principales métodos que debería ofrecer.
4. Dibuje el diagrama de clases UML correspondiente.
5. Explique brevemente por qué el nuevo diseño resulta más fácil de mantener y extender que el diseño original.

Problema 2

Un grupo de hospitales desea desarrollar un sistema sencillo para administrar los médicos que prestan servicios en cada institución.

Cada hospital posee un nombre que lo identifica y dispone de un consultorio principal donde se organizan los profesionales que atienden a los pacientes. El consultorio forma parte del hospital y no tiene sentido que exista de manera independiente.

Los hospitales pueden contratar médicos para que trabajen en la institución. De cada médico interesa conocer únicamente su nombre.

Un mismo médico puede ser contratado por distintos hospitales al mismo tiempo. Por ejemplo, un profesional puede atender algunos días en el Hospital Central y otros días en el Hospital Norte.

Cada vez que un hospital contrata un médico:

- el médico pasa a formar parte del plantel de profesionales del hospital;
- el consultorio principal registra al médico entre los profesionales que atienden allí;
- el médico debe conservar un registro de los hospitales donde trabaja.
- Además, el sistema debe permitir obtener:

- el listado de médicos contratados por un hospital;
- el listado de hospitales donde trabaja un determinado médico.

Actividades

1. Identifique las clases necesarias para modelar el problema.
2. Determine los atributos y métodos principales de cada clase.
3. Identifique las relaciones existentes entre las clases indicando su tipo y justificando brevemente cada decisión.
4. Dibuje el diagrama de clases UML correspondiente.
5. Implemente las clases en Python respetando el diagrama realizado.

Problema 3

Un laboratorio de Bioinformática realiza experimentos biológicos en distintas cámaras de cultivo. Cada cámara cuenta con varios sensores que registran de forma periódica variables ambientales como temperatura, humedad y presión.

Inicialmente, las mediciones son almacenadas en un archivo de texto con el siguiente formato:

```
temperatura,24.5
humedad,68
presion,1020
temperatura,24.8
humedad,72
…
```

Cada experimento posee un conjunto de sensores asociados. Además, distintos investigadores pueden participar simultáneamente en uno o más experimentos.

Cada tipo de sensor debe almacenar las mediciones recibidas y realizar un análisis específico sobre la variable que monitorea:

- El sensor de temperatura debe analizar la evolución de la temperatura a lo largo del experimento e identificar posibles anomalías térmicas.
- El sensor de humedad debe analizar el comportamiento de la humedad e identificar períodos en los que los valores se mantuvieron fuera del rango esperado.
- El sensor de presión debe analizar la evolución de la presión atmosférica e identificar posibles tendencias o variaciones significativas.

Asimismo, todos los sensores deberán generar un resumen estadístico de las mediciones registradas, incluyendo al menos el valor mínimo, máximo y promedio.

El laboratorio desea que el sistema pueda evolucionar para que las mediciones puedan obtenerse desde distintas fuentes de datos y no únicamente desde archivos de texto.

Actividades

a) Análisis y diseño

1. Identifique las clases necesarias para modelar el problema y describa brevemente la responsabilidad de cada una.
2. Determine las relaciones existentes entre las clases, indicando cuáles corresponden a herencia, composición, agregación y asociación.
3. Dibuje el diagrama de clases UML completo, indicando atributos, métodos, visibilidad y multiplicidades.

b) Implementación

1. Implemente las clases respetando el diseño realizado.
2. Utilice una clase abstracta para representar el concepto general de Sensor.
3. Implemente los distintos tipos de sensores utilizando herencia y polimorfismo.
4. Implemente una clase abstracta FuenteDatos, responsable de proporcionar las mediciones al sistema. Desarrolle una implementación concreta denominada LectorArchivo, que obtenga las mediciones desde un archivo de texto.
5. Desarrolle un programa principal que:
    a. cree un experimento;
    b. registre distintos sensores;
    c. lea un archivo de mediciones;
    d. procese la información recibida;
    e. muestre un resumen de los resultados obtenidos por cada sensor.

c) Reflexión

El laboratorio planea reemplazar la lectura del archivo por la recepción de datos desde un broker MQTT.

Explique cómo incorporaría una nueva fuente de datos denominada ClienteMQTT reutilizando el diseño realizado. ¿Qué clases deberían implementarse, cuáles permanecerían sin modificaciones y qué ventajas ofrece esta solución?

Problema 4

Implemente los siguientes patrones de diseño en Python:

A) **Singleton** — Configuración de Aplicación: Implemente una clase Configuracion que garantice una única instancia en toda la aplicación. Debe almacenar parámetros como base_de_datos_url, modo_debug, etc.

B) **Factory Method** — Creación de Documentos: Implemente un sistema que pueda crear diferentes tipos de documentos (PDF, Word, HTML) sin especificar las clases concretas en el código cliente.

C) **Observer** — Sistema de Notificaciones: Implemente un sistema donde un Sujeto (por ejemplo, SensorDeTemperatura) notifique a múltiples Observadores cuando cambie su estado.

D) **Strategy** — Estrategias de Ordenamiento: Implemente un contexto que pueda usar diferentes estrategias de ordenamiento (burbuja, quicksort, mergesort) de forma intercambiable.

Actividad

Responda las siguientes preguntas:

a. ¿En qué situaciones usaría cada patrón?
b. ¿Qué relación existe entre los patrones de diseño y los principios SOLID?