Tecnológico Nacional de México Campus Felipe Carrillo Puerto

Ingeniería en Sistemas Computacionales

Asignatura
Programación lógica y funcional

Tema
Tema 4. Modelo de programación lógica

EVIDENCIA DE APRENDIZAJE
AA 4.2 Reporte practico

Profesor
ING. Paloma Gongora Sabido

Alumno (s):
Wilbert Adrián Góngora Santamaría

(ISC-8A)

Felipe Carrillo Puerto a 05 de JUNIO de 2025

Capturas

Usa una clase `SistemaRiegoInteligente` que almacena parámetros como humedad del suelo, temperatura y nivel de agua.

```python
class SistemaRiegoInteligente:
    def __init__(self):
        # Datos del sistema (podrían venir de sensores en una implementación real)
        self.parametros = {
            'humedad_suelo': 'baja',
            'temperatura': 35,
            'hora': 14,
            'probabilidad_lluvia': False,
            'viento': 'flojo',
            'nivel_tanque': 'suficiente',
            'nivel_fertilizante': 'adecuado'
        }
```

Al iniciarse, ejecuta automáticamente tres análisis clave:
(1) `evaluar_condiciones_riego()` verifica 5 reglas técnicas (como humedad baja y viento moderado), (2) `verificar_alertas()` detecta riesgos como calor extremo o bajo nivel de agua, y (3) `generar_recomendacion()` crea sugerencias personalizada

```python
class SistemaRiegoInteligente:
    def __init__(self):
        # Ejecutar análisis automáticamente al iniciar
        self.analizar_sistema()

    def evaluar_condiciones_riego(self):
        """Evalúa si se cumplen las condiciones para el riego"""
        condiciones = {
            'Humedad del suelo es baja': self.parametros['humedad_suelo'] == 'baja',
            'No hay probabilidad de lluvia': not self.parametros['probabilidad_lluvia'],
            'Hora adecuada (antes de 10 o después de 18)': self.parametros['hora'] < 10 or self.parametros['hora'] > 18,
            'Viento es flojo o moderado': self.parametros['viento'] in ['flojo', 'moderado'],
            'Tanque tiene suficiente agua': self.parametros['nivel_tanque'] == 'suficiente'
        }
        return condiciones

    def verificar_alertas(self):
        """Identifica condiciones que requieren atención especial"""
        alertas = []
```

Finalmente, analizar_sistema() muestra un reporte estructurado con los parámetros actuales, estado de cada condición (✓/✗), conclusiones, alertas y recomendaciones específicas. Este

enfoque orientado a objetos es más modular y escalable que la versión Prolog, permitiendo fácil integración con sensores IoT. El sistema opera automáticamente sin interacción manual, generando decisiones basadas en lógica precisa

```
[Parámetros actuales]
- Humedad Suelo: baja
- Temperatura: 35
- Hora: 14
- Probabilidad Lluvia: False
- Viento: flojo
- Nivel Tanque: suficiente
- Nivel Fertilizante: adecuado

[Evaluación de condiciones para riego]
- Humedad del suelo es baja: Cumple
- No hay probabilidad de lluvia: Cumple
- Hora adecuada (antes de 10 o después de 18): No cumple
- Viento es flojo o moderado: Cumple
- Tanque tiene suficiente agua: Cumple
```

```
- Tanque tiene suficiente agua: Cumple

[Conclusión]: Condiciones no adecuadas para riego

[Alertas activas]
- Temperatura alta (35°C)

[Recomendación]
No se recomienda activar el riego en las condiciones actuales.

[Estado de fertilización]
- Fertilización: Disponible (nivel adecuado)

=== FIN DEL INFORME ===
```

Conclusión

Esta actividad permitió comprender las diferencias fundamentales entre el paradigma lógico de Prolog y el enfoque funcional en Python. Mientras Prolog se basa en la evaluación de predicados y unificación de términos, el enfoque funcional en Python utiliza funciones puras y estructuras de datos inmutables para lograr un comportamiento similar.