INSTRUCTOR 

Solución 02 Notas Kimi

📘 Notas de clase – Divide y vencerás para construcción de código

Autor: Kimi
Objetivo: implementar algoritmos mediante **división de problemas** y **funciones** (recursivas o no).

---

1. Divide y vencerás – construcción de código

Esquema general

1. **Dividir** → descomponer el problema en subproblemas **más pequeños**.
2. **Vencer** → resolver cada subproblema (directamente o recursivamente).
3. **Combinar** → unir las soluciones parciales.

Implementación con funciones

- **Función principal** → inicia el proceso.
- **Funciones auxiliares** → dividen, resuelven y combinan.
- **Recursividad** → opcional, pero **muy útil**.

2. ¿Qué es una función recursiva?

- **Se llama a sí misma** para resolver **subproblemas idénticos.**
- **Caso base** → detiene la recursión.
- **Caso recursivo** → llama a la función con **datos más pequeños.**

Ejemplo:

Python 复制

```python
def factorial(n):
    if n == 0:          # caso base
        return 1
    else:               # caso recursivo
        return n * factorial(n-1)
```

3. ¿Es necesario usar recursión en "divide y vencerás"?

No es obligatorio, pero **es la opción más natural** cuando:

- Los **subproblemas** son **réplicas** del original.
- La **profundidad** de división es **predecible.**

Alternativa: iteración + pila explícita

Python 复制

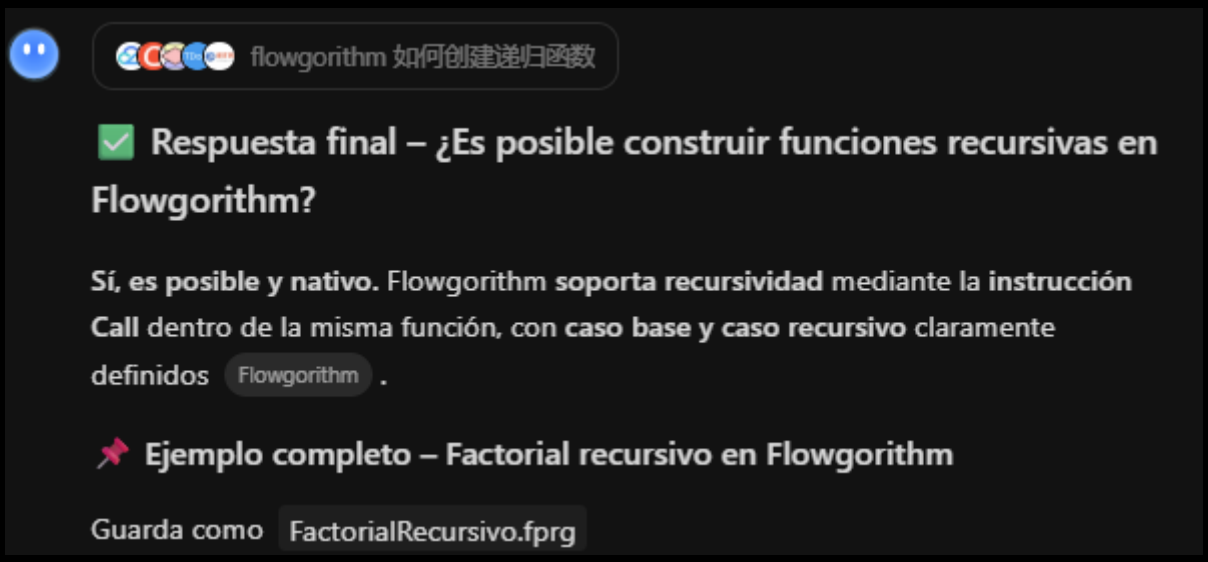
```python
def factorial_iter(n):
    pila = []
    while n > 0:
        pila.append(n)
        n -= 1
    resultado = 1
    while pila:
        resultado *= pila.pop()
    return resultado
```

Conclusión:

- **Puedes usar funciones no recursivas** (iteración + pila).
- **La recursividad es opcional**, pero **más elegante y clara** cuando el problema se **replica.**

INSTRUCTOR

Es posible construir funciones recursivas en flowgorithm




```
<?xml version="1.0"?>
<flowgorithm fileversion="3.0">
  <function name="Factorial" type="Integer">
    <parameter name="n" type="Integer"/>
    <body>
      <if expression="n = 0">
        <then>
          <return value="1"/>
        </then>
        <else>
          <return value="n * Factorial(n - 1)"/>
        </else>
      </if>
    </body>
  </function>

  <function name="Main" type="None">
    <parameter/>
    <body>
      <declare name="n" type="Integer" array="False"/>
      <output text="Ingrese n: "/>
      <input variable="n"/>
      <output text="Factorial = " newline="False"/>
      <output expression="Factorial(n)" newline="True"/>
    </body>
  </function>
</flowgorithm>
```

✅ Conclusión

- **Sí, es posible** construir funciones recursivas en Flowgorithm.
- **Sin límite** de profundidad (hasta stack overflow).
- **Ideal** para: factorial, Fibonacci, Torres de Hanói, divide y vencerás, etc.