Ejercicio 1: **Serie de Fibonacci hasta un Límite**

Escribe un programa que genere los términos de la serie de Fibonacci hasta que el último término sea menor o igual a un número dado por el usuario. La serie de Fibonacci comienza con los términos 0 y 1, y cada término subsiguiente es la suma de los dos términos anteriores.

Instrucciones

1. Pide al usuario un número límite (positivo).
2. Usa un bucle para calcular y mostrar cada término de la serie de Fibonacci hasta que se alcance o supere el límite.
3. Si el número ingresado es menor que 1, muestra un mensaje de error y pide un nuevo número positivo.

Debug: Preguntas

1. **¿Qué sucede si el usuario ingresa un límite de 1? ¿Cuál es el resultado esperado y por qué?**
2. **¿Por qué se imprime a al comienzo del bucle `while` en lugar de al final?**
3. **Si cambiamos la condición del `while` a `numero < limite`, ¿cómo cambia el resultado de la serie de Fibonacci?**

Ejercicio 2: **Número Perfecto**

Un número perfecto es un número entero positivo que es igual a la suma de sus divisores positivos propios, excluyendo el propio número. Escribe un programa que determine si un número ingresado por el usuario es un número perfecto o no.

Instrucciones

1. Pide al usuario un número positivo.
2. Usa un bucle para encontrar todos los divisores propios del número (excluyendo el número mismo).
3. Calcula la suma de los divisores y determina si es igual al número original.
4. Imprime un mensaje indicando si el número es perfecto o no.

Debug: Preguntas

1. **¿Qué valor tendría `sumaDivisores` si se ingresa el número 6? ¿Por qué es considerado perfecto?**

2. **¿Qué cambios serían necesarios para que el programa también identifique los números abundantes (cuando la suma de divisores es mayor que el número) y deficientes (cuando la suma es menor que el número)?**
3. **¿Por qué el bucle `for` solo llega hasta `numero - 1` en lugar de incluir `numero`?**

Ejercicio 3: **Descomposición en Factores Primos**

Escribe un programa que descomponga un número entero positivo en sus factores primos. Por ejemplo, para el número 18, el programa debe mostrar `18 = 2 * 3 * 3`.

Instrucciones

1. Pide al usuario un número positivo.
2. Usa un bucle `while` para dividir el número por el menor divisor posible (empezando desde 2).
3. Cada vez que encuentres un divisor, guarda el divisor en una lista y divide el número.
4. Muestra los factores encontrados como un producto de números primos.

Debug: Preguntas

1. **¿Qué ocurre en el programa cuando el número ingresado es primo? ¿Qué valores se imprimen en este caso?**
2. **Si cambiamos el valor inicial de `divisor` a 3 en lugar de 2, ¿cómo afecta esto a la salida?**
3. **¿Cómo podrías modificar el programa para que también funcione para números negativos (mostrando el signo `-` en la descomposición final)?**