Guía de Estudio: Pensamiento Analítico-Lógico y Programación

Esta guía se basa únicamente en los temas presentes en los archivos proporcionados. No contiene referencias a preguntas, sino que explica los conocimientos necesarios para poder responder un banco de reactivos de pensamiento lógico y programación.

1. Pensamiento lógico y razonamiento computacional

• Comprender qué es el pensamiento lógico en programación: analizar problemas, reconocer patrones y estructurar soluciones paso a paso.
• Diferencia entre programar y escribir código: programar implica pensar; el código es solo la traducción del pensamiento.
• Papel de la deducción y el pensamiento crítico: cómo se deriva una conclusión válida a partir de premisas.

2. Algoritmos y representación de procesos

• Qué es un algoritmo: pasos ordenados y finitos para resolver un problema.
• Representaciones: pseudocódigo y diagramas de flujo.
• Secuencias: cómo interpretar listas de instrucciones, orden, inicio, fin y rutas de decisión.

3. Patrones, series y secuencias

• Identificación de patrones numéricos (duplicación, progresiones, Fibonacci).
• Series alfabéticas y cambios de posición.
• Cálculo de días de la semana mediante aritmética modular.
• Analogías: relaciones entre conceptos (software–hardware, causa–efecto, clasificación).

4. Lógica proposicional y operadores

• Comprender relaciones: mayor que, menor, igual, implicaciones y conclusiones válidas.
• Operadores lógicos: AND, OR, NOT.
• Condiciones en programación: evaluación verdadera/falsa.
• Identificar falacias o conclusiones incorrectas.

5. Variables y tipos de datos

• Qué es una variable: contenedor de información que puede cambiar.
• Tipos comunes: enteros, cadenas, booleanos.
• Uso de variables booleanas para controlar decisiones en el programa.

6. Estructuras de control: condicionales y ciclos

• Condicionales if/else: cómo funcionan, qué ocurre cuando una condición es verdadera o falsa.
• Ciclos: while, for, do-while; repetición controlada y condición de parada.
• Trazado mental de código: seguir la ejecución paso a paso.

7. Análisis y depuración de código

• Identificar errores lógicos: cuando el programa corre pero produce un resultado incorrecto.
• Depuración: seguir el flujo del programa, revisar valores e instrucciones.
• Técnicas como "rubber duck debugging": explicar el proceso paso a paso para encontrar fallas.

8. Estructuras de datos básicas

• Arreglos: índices, longitud, acceso a elementos.
• Pilas (LIFO) y colas (FIFO): conceptos y usos.
• Validación de entradas: evitar errores por datos incorrectos.

9. Complejidad y eficiencia

• Concepto de eficiencia: resolver con menos recursos y tiempo.
• Notación Big O: O(1), O(n), O(log n).
• Búsqueda binaria vs. fuerza bruta.
• Cuándo considerar eficiencia en un algoritmo.

10. Conceptos de hardware y sistemas

• Memoria RAM: almacenamiento temporal de programas en ejecución.
• Diferencia hardware vs. software.
• Qué significa que un sistema sea eficiente.