Taller de Consulta de Conceptos Básicos en Lógica de Programación

1. Introducción a la lógica de programación:

a. ¿Qué es la lógica de programación y por qué es importante en el desarrollo de software?

Es un tipo de paradigma de programación declarativa. El resto de los sub-paradigmas de programación dentro de la programación declarativa son: programación funcional, programación con restricciones, también es importante ya que es posible aprender a escribir un código para que la computadora interprete correctamente

b. ¿Cuáles son los beneficios de tener una buena comprensión de la lógica de programación?

La lógica en la programación es esencial para desarrollar software de calidad y eficiente. Para mejorar nuestra lógica de programación, debemos pensar de forma resolutiva, practicar, aprender algoritmos, estudiar diferentes paradigmas de programación

2. Algoritmos:  
a. ¿Qué es un algoritmo y cómo se relaciona con la lógica de programación?

El algoritmo es la propia secuencia de instrucciones para la ejecución de una tarea. De una forma más simple, el algoritmo es una receta o ruta que indica todos los procedimientos necesarios para realizar algo o resolver un problema  
b. ¿Cuáles son los elementos básicos de un algoritmo? (Secuencia, selección y repetición)

Todo algoritmo consta de tres secciones principales: Entrada: Es la introducción de datos para ser transformados. Proceso: Es el conjunto de operaciones a realizar para dar solución al problema. Salida: Son los resultados obtenidos a través del proceso  
c. Ejemplos de algoritmos sencillos para diferentes situaciones (por ejemplo, cálculos matemáticos, toma de decisiones, bucles).

Los bucles permiten que se ejecute repetidamente un conjunto de instrucciones, bien un número predeterminado de veces, o bien hasta que se verifique una determinada condición. En términos de un lenguaje de programación, que se verifique o no una condición se traduce en que una (adecuada) expresión lógica tome el valor VERDADERO (TRUE) o tome el valor FALSO (FALSE). En los casos más sencillos y habituales la condición suele ser una comparación entre dos datos, como, por ejemplo: si a < b hacer una cosa y en caso contrario hacer otra distinta  
  
3. Estructuras de control:  
a. Estructuras de control secuenciales: ¿cómo se ejecutan las instrucciones en secuencia?

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso  
b. Estructuras de control condicionales: ¿cómo se toman decisiones en la programación? (if, if-else, switch)

Las construcciones if, else y else-if, permite condicionar la ejecución de uno o varios bloques de sentencias al cumplimiento de una o varias condiciones, sirve para saber si o y si no

c. Estructuras de control de bucle: ¿cómo se repiten las instrucciones? (while, do-while, for)

No debemos confundir la representación gráfica de la estructura repetitiva Do While ... Loop (Mientras) con la estructura condicional If (Si)

Funcionamiento: En primer lugar, se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero.  
A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva.  
En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

4. Variables y tipos de datos:  
a. ¿Qué son las variables y cómo se utilizan en la programación?

Una variable es donde se almacenan y se recuperan los datos de un programa. Así de simple. En programación, la utilizamos para guardar datos y estados, asignar ciertos valores de variables a otras, representar valores de expresiones matemáticas y mostrar valores por pantallas  
b. Tipos de datos básicos: enteros, números de punto flotante, cadenas de texto, booleanos.

-Booleano (p. Ej., verdadero o falso)

-Carácter (p. Ej., a)

-Fecha (p. Ej., 01/03/2016)

-Doble (p. Ej., 1.79769313486232E308)

-Número de punto flotante (p. Ej., 1.234)

-Entero (p. Ej., 1234)

-Largo [Long] (p. Ej., 123456789)

-Corto [Short] (por ejemplo, 0)

-Cadena [String] (por ejemplo, abcd)

-Void (p. Ej., Sin datos)

c. Declaración y asignación de variables.

Una variable se puede declarar con el ';' al final o sin él. La diferencia es que con el ';', la variable no es mostrada en el terminal. Si queremos darle un nuevo valor a una variable solo hay que declarar el nuevo valor que va a tener la variable como se ha hecho anteriormente.  
d. Conversión de tipos de datos.

Una conversión de tipos es una manera de informar explícitamente al compilador de que se pretende realizar la conversión y se es consciente de que se puede producir pérdida de datos o la conversión de tipos puede fallar en tiempo de ejecución  
  
5. Funciones:  
a. ¿Qué es una función y por qué son útiles?

Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función. La siguiente función acepta dos enteros de un autor de llamada y devuelve su suma; a y b son parámetros de tipo int  
b. Definición y llamada de funciones.

Una función es un conjunto de sentencias con nombre exclusivo que realizan una determinada tarea. Una función puede tomar un conjunto de argumentos sobre los que operar y puede devolver un valor que represente el resultado de la tarea que ha realizado

c. Parámetros y argumentos de función.

Parámetro es el nombre que recibe una de las variables que aparecen en la declaración de la función. Argumento es el nombre que recibe uno de los valores que es suministrado a la función cuando es invocada. Un argumento puede ser un valor literal, un valor calculable o unavariable  
d. Retorno de valores de función.  
Generalmente, se usa un valor de retorno donde la función es un paso intermedio en un cálculo de algún tipo. Quieres llegar a un resultado final, que involucra algunos valores. Esos valores deben ser calculados por una función, que luego devuelve los resultados para que puedan usarse en la siguiente etapa del cálculo  
6. Resolución de problemas:  
a. Enfoque paso a paso para resolver problemas de programación.

comprender el problema, diseñar un plan,dividir,codificar,depurar y probar,documentar,revisar y repetir  
b. Descomposición de problemas en subproblemas más pequeños.

analizar el problema, identificar las tareas principales, dividir las tareas, integra las soluciones de los sub problemas  
c. Uso de pseudocódigo y diagramas de flujo para planificar soluciones.

El uso de pseudocódigo y diagramas de flujo es una práctica común y útil para planificar y diseñar soluciones de programación antes de comenzar a escribir el código real. Estas herramientas te permiten visualizar y organizar la lógica de tu programa de una manera clara y comprensible  
  
7. Recursos adicionales:  
a. Proporciona una lista de recursos en línea (tutoriales, cursos, libros) para profundizar en la lógica de programación.

Cursos en línea

odecademy (https://www.codecademy.com/): Ofrece cursos interactivos gratuitos y pagados sobre diferentes lenguajes de programación, incluida la lógica de programación.

Coursera (https://www.coursera.org/): Proporciona una amplia gama de cursos en línea, muchos de los cuales se centran en la lógica de programación y algoritmos.

edX (https://www.edx.org/): Ofrece cursos de universidades de renombre en todo el mundo, con una amplia selección de cursos relacionados con la lógica de programación

Libros

Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People" de Aditya Bhargava.

"Introduction to the Theory of Computation" de Michael Sipser.

"Algorithm Design Manual" de Steven S. Skiena