CURSO DE LÓGICA DE PROGRAMACIÓN

INTRODUCCIÓN

Hoy en día es esencial saber programar. La programación está en todas partes: en las apps que usamos a diario como Instagram o TikTok, en los coches autónomos e incluso en las neveras inteligentes que nos avisan cuando un alimento está a punto de caducar. Pero para que todo esto funcione, no basta con conocer un lenguaje de programación: es necesario aprender a darle a la máquina las instrucciones correctas de forma clara y ordenada. En este curso nos centraremos en cómo estructurar esas instrucciones. Porque, cuando entiendes el problema y sabes descomponerlo en pasos lógicos, escribir el código se vuelve mucho más sencillo.

Este curso está organizado en 6 módulos diseñados para llevarte de cero a un pensamiento lógico y estructurado en programación. No necesitas conocimientos previos: empezaremos desde los fundamentos más básicos y avanzaremos paso a paso, de forma clara y práctica. Cada módulo te ayudará a desarrollar nuevas habilidades, desde aprender a plantear problemas hasta diseñar tus propios algoritmos. La idea es que, al finalizar el curso, te sientas capaz de entender y aplicar la lógica de programación sin importar el lenguaje que elijas después.

Eso sí, la práctica es fundamental. Igual que sucede cuando aprendes un idioma o entrenas en el gimnasio: si dejas de practicar, la habilidad se oxida. Por eso, a lo largo del curso te propondré ejercicios y retos que mantendrán tu mente en forma y harán que tu progreso sea real y duradero. En cada módulo habrá tanto ejemplos guiados como ejercicios propuestos. No hará falta que escribas código para resolverlos, aunque si ya conoces algún lenguaje de programación, también encontrarás las soluciones en Python para que puedas practicarlas de forma opcional

MÓDULO 1: PENSAMIENTO ALGORÍTMICO

¿Qué es la lógica de programación?

A todos nos ha pasado que alguien nos diga algo que no tiene ningún sentido. Es como si te dijera ahora que el cielo está hecho de ladrillos. Absurdo, ¿verdad? Pues algo parecido ocurre en programación: cuando le damos instrucciones a un ordenador, no siempre las interpreta como esperamos. Por eso es clave aprender a comunicarnos con él de manera clara y precisa, y en este módulo vamos a empezar a descubrir cómo hacerlo.

Para saber si estamos dándole instrucciones adecuadas, podemos apoyarnos en un esquema que muestre la secuencia de pasos a seguir. Es como cuando usamos Google Maps para llegar a un destino: la aplicación nos indica por dónde girar y qué salida tomar en cada rotonda. Un ejemplo más técnico: para saber si un número es par o impar, primero necesitamos conocer el

número. Sería absurdo intentar comprobarlo antes de tener ese dato, ¿verdad? A continuación tienes el código en Python. No es necesario que lo entiendas por completo, pero si ya conoces algo de programación, puedes analizarlo para ver cómo se traduce la lógica en código.

```python
valor = int(input("Ingrese un valor: "))
if valor % 2 == 0:
    print(f"El valor {valor} es par")
else:
    print(f"El valor {valor} es impar")
```

Concepto de algoritmo

Cuando vamos a la compra y queremos pan, nos dirigimos al pasillo de bollería y buscamos nuestro pan habitual. Pero no está. Entonces aparecen dos opciones: uno más caro pero más saludable, y otro más barato aunque menos saludable. Finalmente, eliges el saludable. Aunque no lo parezca, ahí ya hemos seguido un algoritmo: un conjunto de pasos y decisiones para llegar a una solución. Otro ejemplo cotidiano es el de atarse los zapatos: si son de cuerda aplicamos una técnica, y si son de velcro, otra distinta. Este concepto es esencial en la programación, porque la lógica de los algoritmos es la base que sostiene este mundo.

Los algoritmos sirven para indicarle al ordenador la secuencia exacta de instrucciones que debe seguir. A diferencia de los humanos, que podemos deducir cosas por contexto (por ejemplo, entender que 'coge la taza' implica buscarla en la mesa si no está en la mano), el ordenador no lo hará. Por eso es fundamental explicarle cada paso de forma clara y precisa.

Pensemos en otro ejemplo: si queremos saber el valor en una posición cualquiera de la sucesión de Fibonacci, ¿cómo lo calcularíamos? Sabemos que esta sucesión se define como la suma de los dos valores anteriores y que empieza con 0 y 1. A partir de esa definición, podemos obtener cualquier valor de la secuencia siguiendo ese patrón. A continuación, te dejo el código por si lo quieres analizar:

```python
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

num = int(input("Ingrese un número: "))
print(f"El número Fibonacci de {num} es: {fibonacci(num)}")
#El 0 está en la posición 0, el 1 en la posición 1, etc.
```

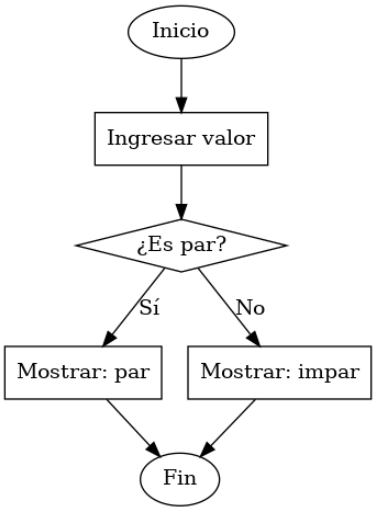
En el ejemplo anterior, se ha definido la secuencia de Fibonacci por separado del resto del código. No te preocupes por entenderlo a fondo ahora: más adelante veremos cómo empaquetar estas instrucciones en algo llamado función, que nos permitirá reutilizar el mismo algoritmo sin tener que escribirlo cada vez. En este ejemplo de Fibonacci hemos escrito las instrucciones paso a paso, pero pronto aprenderás una forma más elegante y reutilizable de hacerlo.

Introducción a diagramas de flujo

¿Cómo podemos diseñar dichos algoritmos? Una de las herramientas más útiles son los diagramas de flujo. Estos diagramas nos permiten visualizar de manera clara todas las posibles rutas y decisiones que puede tomar nuestro programa, lo que facilita entender y planificar la lógica antes de escribir una sola línea de código.

Para estructurar un diagrama de flujo debemos pensar primero en el objetivo del algoritmo y luego desglosar las acciones en pasos simples y lógicos. Todo diagrama empieza con un símbolo de inicio y termina con un símbolo de fin, conectados por flechas que indican el orden de ejecución. Cada acción concreta, como 'sumar dos números', se representa con un rectángulo, mientras que las decisiones o bifurcaciones, como '¿es el número par?', se muestran en un rombo que permite distintas rutas según la respuesta. La clave está en que cada paso sea claro y que el flujo siempre avance de manera lógica y sin ambigüedades. Antes de programar, este esquema te ayuda a detectar errores y a asegurarte de que tu solución tiene sentido.

Veamos un ejemplo sencillo de cómo estructurar un diagrama de flujo usando el caso de los números pares e impares. Primero, pedimos al usuario que introduzca un valor numérico. Luego, calculamos el resto de dividir ese número entre 2. A continuación, comprobamos: si el resto es igual a 0, el número es par; en caso contrario, es impar. Este flujo se puede representar fácilmente en un diagrama con un inicio, una entrada de datos, un proceso de cálculo, una decisión y, finalmente, dos posibles salidas.



Ejercicios: Diseñar algoritmos para tareas del día a día

MÓDULO 2: ESTRUCTURAS DE CONTROL

MÓDULO 3: PENSAMIENTO RECURSIVO Y MODULARIDAD

MÓDULO 4: ESTRUCTURAS DE DATOS LÓGICOS

MÓDULO 5: ALGORITMOS CLÁSICOS Y ANÁLISIS CLÁSICOS

MÓDULO 6: PENSAMIENTO COMPUTACIONAL Y PROBLEMAS ABIERTOS