

3. ¡Eureka: hemos descubierto los algoritmos!

Un paso más en nuestro camino por encontrar la solución al problema que se nos hayan planteado es redefinir la metodología del profesor Pólya. Por tanto, ahora nuestros **planes** se convertirán en **algoritmos**. Es decir, trazaremos el plan diseñando un algoritmo. Traduciremos el problema a través de un lenguaje de programación (en nuestro caso con **Java**) y llegaremos a la revisión final con un programa depurado y listo para ejecutarlo.

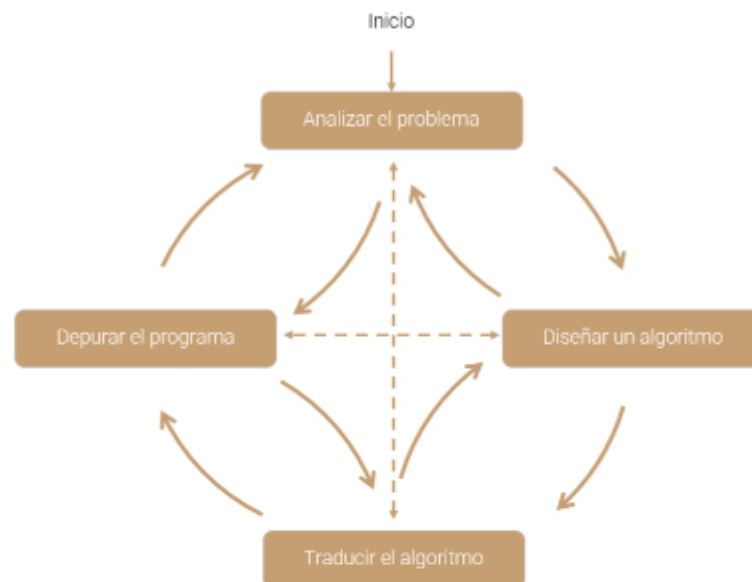


Figura 1.7. ¿Cómo resuelvo un problema mediante algoritmos y programas?

Pero empecemos nuevamente por el inicio. En la primera fase, a la hora de **analizar**, se debe **entender** el problema, por lo que debemos:

- Reformular el problema de la manera más sencilla posible.
- Saber si existen restricciones del problema.
- Identificar los datos de entrada que es preciso utilizar.
- Definir las operaciones por realizar.
- Conocer el resultado esperado, es decir, el dato de salida.



IMPORTANTE

Según decía Steve Jobs: «Decidir qué no hacer es tan importante como decidir qué hacer». Por tanto, céntrate solo en lo importante y fundamental para solucionarlo.

En la segunda fase, al **diseñar** el algoritmo, se debe realizar la representación gráfica o textual mediante un **diagrama** o un **pseudocódigo** de la secuencia de las operaciones de forma lógica para encontrar la solución.

Los algoritmos se pueden representar de los siguientes modos:

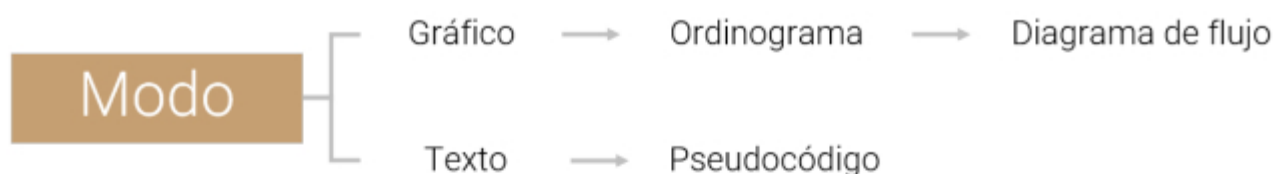


Figura 1.8. Diferentes formas de definir un algoritmo

Vamos a aprender cómo empezar a utilizarlos. Comenzaremos con las ordenes básicas, así que, en el caso práctico ampliado de este apartado, añadiremos el resto de los símbolos utilizados.

3.1. Ordinograma

Un ordinograma o un diagrama para diseñar un algoritmo es conocido como **diagrama de flujo**. Este representa la secuencia lógica de nuestro análisis, cuya simbología más básica es esta:





Símbolo	Significado
	Terminal. Representa el inicio o el fin de un programa. Solo debe haber uno de cada.
	Entrada/Salida. Nos permite introducir datos con un periférico (como un teclado), así como mostrarlos por pantalla.
	Proceso. Acciones del programa que procesan los datos del sistema.
	Línea de flujo. Indica la dirección de ejecución del algoritmo. Siempre irá dirigido en sentido descendente. Podemos ampliar el diagrama con alguna bifurcación lateral a izquierda o a derecha.

Tabla 1.1. Símbolos elementales de un diagrama de flujo

El ejemplo más básico de representación del flujo de datos de un proceso consta de: un inicio y un final, datos de entrada, instrucciones por realizar y un dato de salida. Por ejemplo, para calcular el perímetro de una circunferencia, haríamos lo siguiente:

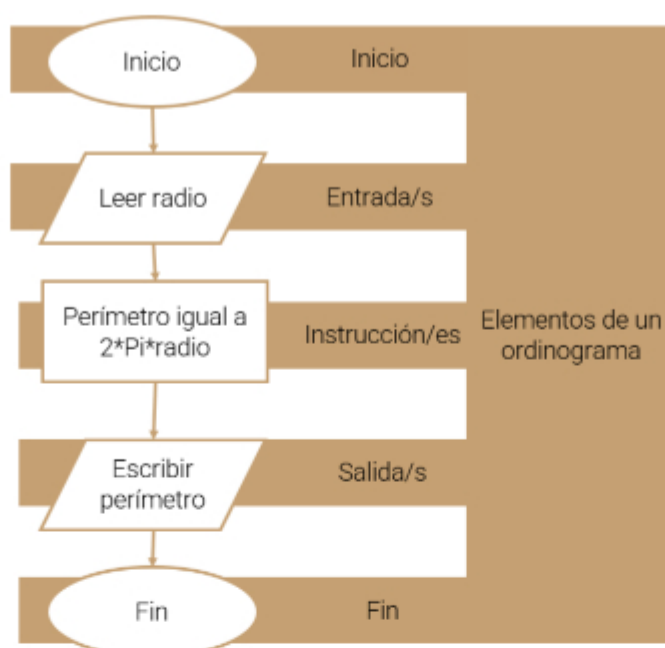


Figura 1.9. Ejemplo de diagrama de flujo para el cálculo del perímetro de un círculo.

3.2. Pseudocódigo

Vamos a ver cómo se crearía un pseudocódigo escrito en lenguaje natural. Este permite escribir las instrucciones que conducen a la resolución del problema utilizando estructuras de programación. Sus reglas son:

- Cada instrucción se escribe en una línea. La primera es la de **inicio** y la última, la de **fin**.
- El conjunto de palabras reservadas se escribe en minúsculas: **leer**, **asignar**, **escribir** u operaciones como **sumar**, **restar**, etcétera.
- Código sangrado o indentado.

El caso anterior se expresaría en pseudocódigo de esta manera:

Programa: Circunferencia				Cabera del algoritmo
Módulo: Calcular Perímetro				
Inicio	Datos	Constantes	Pi: Número real = 3,1416	Zona o bloque de datos
		Variables	Radio: número real Perímetro: número real	
	Algoritmo	Leer radio Perímetro igual a 2*Pi*radio Escribir perímetro		Zona o bloque de acciones
Fin				Cuerpo del algoritmo

Figura 1.10. Ejemplo de un pseudocódigo para el cálculo del perímetro de un círculo.

Una vez resuelto con nuestro algoritmo, en la tercera etapa lo traduciremos a un **programa**, donde trasladaremos esas órdenes del algoritmo a unas instrucciones de un lenguaje de programación concreto (como podría ser C, C++, C#, **Java** o Python, entre otros).

Y por último, en la cuarta etapa, habrá que ir **depurando** nuestras instrucciones escritas en el lenguaje de programación elegido, para verificar que todo está correctamente escrito y que funciona. Solo así habremos conseguido obtener finalmente nuestro **programa, capaz de** resolver el problema inicial que deseábamos solucionar dentro de un sistema informático.

