## **Método Polya-EPCER**

## **Introducción: El Miedo a la Pantalla en Blanco**

Todo estudiante de programación ha sentido esto: te enfrentas a un nuevo problema, abres tu editor de código y… te quedas mirando una pantalla en blanco. ¿Por dónde empezar? ¿Qué escribir primero? Esta parálisis es común, pero superable.

Para combatir esto, no necesitamos un truco de magia, sino un **método**: una estrategia probada que nos guíe desde el enunciado del problema hasta una solución funcional y robusta. Aquí es donde entra en juego el trabajo del matemático **George Polya**.

Polya desarrolló un método de cuatro pasos para resolver problemas matemáticos que es tan efectivo que puede adaptarse perfectamente al mundo de la programación. Esta guía te presentará una versión adaptada de su método en cinco pasos, diseñada específicamente para ayudarte a escribir mejor código en C y Python.

## **El Método Polya Adaptado a la Programación**

Hemos ajustado los pasos clásicos de Polya para que se alineen con el ciclo de vida del desarrollo de software. Nuestro método consta de cinco fases claras y lógicas.

### **1. Entender el Requerimiento 🤔**

**Objetivo:** Saber exactamente qué se te pide hacer. No puedes resolver un problema si no lo entiendes a la perfección. Invertir tiempo aquí te ahorrará horas de frustración más adelante.

**Actividades Sugeridas:**

* **Lee el problema varias veces.** La primera lectura es para tener una idea general; las siguientes son para captar los detalles.
* **Subraya los sustantivos y los verbos clave.** Los sustantivos suelen ser las variables o estructuras de datos (ej: “área”, “figura”, “lado”) y los verbos son las acciones o funciones (ej: “calcular”, “imprimir”, “pedir”).
* **Hazte preguntas:**
  + ¿Qué información me dan (entradas)?
  + ¿Qué debo producir (salidas)?
  + ¿Hay condiciones o reglas especiales? (ej: “el lado debe ser positivo”).
* **Explica el problema con tus propias palabras.** Si puedes explicárselo a un compañero (o a un patito de hule), es señal de que lo has entendido.

### **2. Planificar la Solución 📝**

**Objetivo:** Diseñar un “mapa” o “plano” de tu programa antes de escribir una sola línea de código. La clave es pensar en la lógica, no en la sintaxis de C o Python.

**Actividades Sugeridas:**

* **Define las entradas y salidas:** Lista qué datos necesitas pedirle al usuario y qué datos le vas a mostrar como resultado.
* **Escribe pseudocódigo:** Es una descripción de los pasos de tu programa en lenguaje natural, pero con una estructura similar a la del código. Es tu plan de acción.
* **Dibuja diagramas de flujo:** Si el problema involucra decisiones complejas o bucles, un diagrama puede ayudar a visualizar la lógica.
* **Divide el problema:** Si es un problema grande, divídelo en sub-problemas más pequeños y manejables. Cada sub-problema puede convertirse en una función.

### **3. Codificar lo Planificado 💻**

**Objetivo:** Traducir tu plan (el pseudocódigo o diagrama) al lenguaje de programación elegido (C o Python).

**Actividades Sugeridas:**

* **Sigue tu plan paso a paso.** No intentes improvisar. Si tu plan es sólido, esta fase debería ser una traducción casi mecánica.
* **Escribe código limpio y legible:** Usa nombres de variables descriptivos (ej: area\_cuadrado en lugar de ac), comenta las partes complejas y usa una indentación consistente.
* **Codifica por partes:** Si dividiste el problema, codifica y prueba cada función por separado antes de unirlas.

### **4. Correr y Probar el Código ▶️**

**Objetivo:** Verificar que tu código funciona como se esperaba. No basta con que el programa “corra” sin errores; debe producir el resultado correcto para todos los casos posibles.

**Actividades Sugeridas:**

* **Pruebas con casos “felices”:** Usa entradas típicas y esperadas para ver si el resultado es el correcto. (Ej: si calculas el área de un cuadrado, prueba con un lado de 4).
* **Pruebas con casos límite (Edge Cases):** ¿Qué pasa en los extremos? Prueba con ceros, números muy grandes o, si aplica, valores negativos. (Ej: ¿qué hace tu programa si el lado es 0? ¿Y si es -5?).
* **Pruebas con entradas inválidas:** ¿Qué pasa si el usuario introduce texto cuando se espera un número? Tu programa no debería “romperse”.

### **5. Reflexionar y Refactorizar 🧐**

**Objetivo:** Revisar tu solución final y pensar en cómo podría mejorarse. Un buen programador no se conforma con “funciona”, sino que busca que sea “eficiente, legible y mantenible”.

**Actividades Sugeridas:**

* **Revisa tu código:** ¿Hay partes repetitivas que podrías convertir en una función? ¿Son claros los nombres de las variables? ¿Podrías hacerlo más simple?
* **Piensa en la eficiencia:** ¿Hay alguna operación innecesaria que esté ralentizando tu programa? (Para problemas simples esto no es crítico, pero es un buen hábito).
* **Considera la escalabilidad:** ¿Sería fácil añadir una nueva funcionalidad? Por ejemplo, si tu programa calcula el área de un cuadrado, ¿qué tan difícil sería añadir el cálculo para un rectángulo o un círculo?
* **Aprende de tus errores:** Si tuviste un bug difícil de encontrar, entiende por qué ocurrió para no volver a cometer el mismo error.

## **Ejemplo Práctico: Calculadora de Áreas Geométricas**

Vamos a aplicar el método para crear un programa que calcule el área de un cuadrado.

#### **Paso 1: Entender el Requerimiento**

* **Requerimiento:** Crear un programa que calcule el área de un cuadrado.
* **Entrada:** La longitud de un lado del cuadrado.
* **Salida:** El área calculada del cuadrado.
* **Condiciones:** El área es el lado elevado al cuadrado (Area = lado^2). Implícitamente, la longitud del lado no puede ser negativa.

#### **Paso 2: Planificar la Solución**

* **Pseudocódigo:**
  + Mostrar un mensaje de bienvenida.
  + Pedir al usuario que introduzca la longitud del lado.
  + Almacenar ese valor en un variable llamada “lado”.
  + Calcular el área: area = lado \* lado.
  + Mostrar el resultado del area al usuario.

#### **Paso 3: Codificar lo Planificado**

Se traduce el pseudocódigo a C y Python.

**Código en Python:**

**# 1. Mensaje de bienvenida**  
**print("Calculadora de Área de un Cuadrado")**  
  
**# 2. Pedir la entrada**  
**lado\_str = input("Introduce la longitud del lado: ")**  
  
**# 3. Almacenar y convertir**  
**lado = float(lado\_str)**  
  
**# 4. Calcular el área**  
**area = lado \* lado**  
  
**# 5. Mostrar el resultado**  
**print(f"El área del cuadrado es: {area}")**

**Código en C:**

**#include <stdio.h>**  
  
**int main() {**  
 **// 1. Mensaje de bienvenida (implícito en el printf siguiente)**  
 **float lado, area;**  
  
 **// 2 y 3. Pedir y almacenar la entrada**  
 **printf("Introduce la longitud del lado: ");**  
 **scanf("%f", &lado);**  
  
 **// 4. Calcular el área**  
 **area = lado \* lado;**  
  
 **// 5. Mostrar el resultado**  
 **printf("El área del cuadrado es: %.2f\n", area);**  
  
 **return 0;**  
**}**

#### **Paso 4: Correr y Probar el Código**

* **Caso feliz:** Entrada: 5. Salida esperada: 25. Ambos programas funcionan.
* **Caso límite:** Entrada: -4. Salida: 16. El cálculo es matemáticamente correcto, pero un área no puede venir de un lado negativo. ¡Esto es un problema conceptual! Nuestro programa funciona, pero no es robusto.

#### **Paso 5: Reflexionar y Refactorizar**

* **Reflexión:** El programa falla conceptualmente con números negativos. Deberíamos validar la entrada. Además, solo sirve para cuadrados. ¿Podríamos hacerlo más versátil?
* **Acción de Refactorización:** Añadir una validación y expandir para incluir un rectángulo.

**Código Mejorado en Python:**

def calcular\_area\_cuadrado():  
 lado\_str = input("Introduce la longitud del lado del cuadrado: ")  
 lado = float(lado\_str)  
 if lado >= 0:  
 area = lado \* lado  
 print(f"El área del cuadrado es: {area}")  
 else:  
 print("Error: La longitud del lado no puede ser negativa.")  
  
def calcular\_area\_rectangulo():  
 base\_str = input("Introduce la base del rectángulo: ")  
 altura\_str = input("Introduce la altura del rectángulo: ")  
 base = float(base\_str)  
 altura = float(altura\_str)  
 if base >= 0 and altura >= 0:  
 area = base \* altura  
 print(f"El área del rectángulo es: {area}")  
 else:  
 print("Error: Las dimensiones no pueden ser negativas.")  
  
# Programa principal  
print("Calculadora de Áreas")  
opcion = input("Elige la figura (1: Cuadrado, 2: Rectángulo): ")  
  
if opcion == '1':  
 calcular\_area\_cuadrado()  
elif opcion == '2':  
 calcular\_area\_rectangulo()  
else:  
 print("Opción no válida.")

Esta nueva versión es más **robusta** (valida entradas) y **escalable** (es fácil añadir un elif opcion == '3': para otra figura).

## **Ventajas y Desventajas del Método**

### **Ventajas 👍**

* **Reduce el estrés:** Te da un camino claro a seguir, eliminando la ansiedad de la “página en blanco”.
* **Mejora la calidad del código:** La planificación conduce a un software mejor estructurado, más legible y con menos errores.
* **Facilita la depuración:** Es más fácil encontrar errores en un plan lógico o en un código bien organizado que en un “código espagueti”.
* **Ahorra tiempo a largo plazo:** Aunque parezca que planificar es lento, evita tener que reescribir grandes partes del código por no haber pensado bien el problema al inicio.

### **Desventajas 👎**

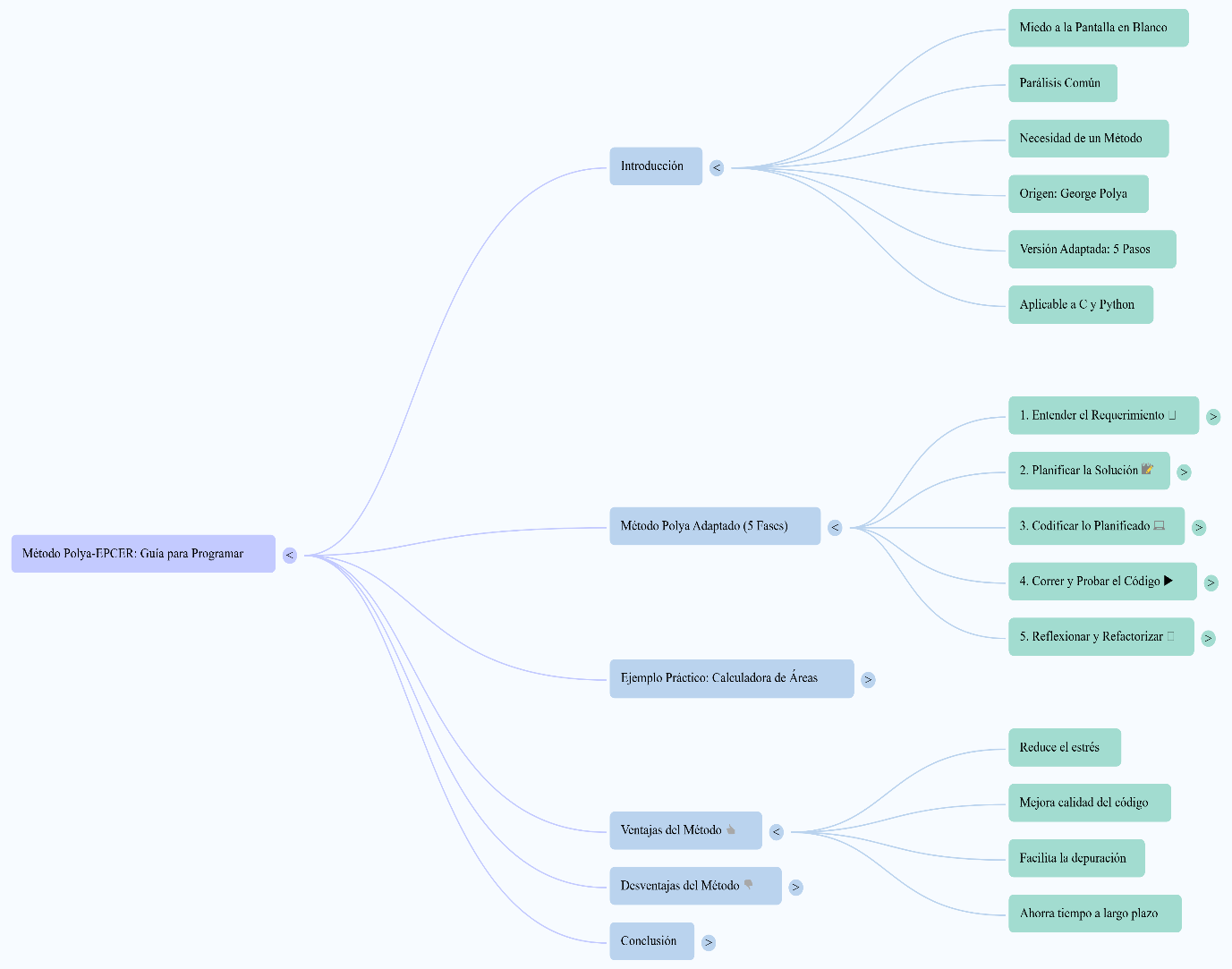
* **Puede parecer lento al principio:** Para problemas muy simples, seguir todos los pasos puede sentirse como un exceso de burocracia.
* **Requiere disciplina:** Es tentador saltar directamente a la codificación. Se necesita práctica para hacer de la planificación un hábito.

## **Conclusión**

El método Polya-EPCER no es una fórmula mágica, sino una herramienta de pensamiento. Al principio, te animamos a seguir los cinco pasos de manera explícita, incluso para los problemas más sencillos. Con la práctica, estos pasos se volverán una segunda naturaleza, una forma intuitiva de abordar cualquier desafío de programación, sin importar su complejidad.

**¡Feliz codificación**

**Mapa conceptual del Método Polya-EPCER**



**Ejemplo Codificación de un programa**

/\*

**Nombre del Programa:** Calculador de Áreas de figuras geométricas

**Descripción**: Este programa calcula el área de un cuadrado, rectángulo, triángulo y círculo.

**Fórmulas utilizadas:**

* Cuadrado: área = lado \* lado
* Rectángulo: área = base \* altura
* Triángulo: área = (base \* altura) / 2
* Círculo: área = π \* radio²

**Programador**: Prof. Felipe Martínez

**Fecha: 2023-11-15**

**\*/**

**/\***

**Plan de codificación:**

1 Declarar variables

fm\_lado\_cuadrado

fm\_base\_rectangulo

fm\_altura\_rectangulo

fm\_base\_triangulo

fm\_altura\_triangulo

fm\_radio\_circulo

2 Para cada figura geométrica

Leer lados o radio

Calcular el área

Imprimir el área calculada

**\*/**

**/\* Código del programa principal**

/\* #include <stdio.h> #include <math.h>

int main() {

// Variables para el cuadrado   
float fm\_lado\_cuadrado;   
float fm\_area\_cuadrado;

// Variables para el rectángulo  
float fm\_base\_rectangulo;  
float fm\_altura\_rectangulo;  
float fm\_area\_rectangulo;

// Variables para el triángulo  
float fm\_base\_triangulo;  
float fm\_altura\_triangulo;  
float fm\_area\_triangulo;  
  
// Variables para el círculo  
float fm\_radio\_circulo;  
float fm\_area\_circulo;  
  
// Solicitando datos para el cuadrado  
printf("Ingrese el lado del cuadrado: ");  
scanf("%f", &fm\_lado\_cuadrado);  
fm\_area\_cuadrado = fm\_lado\_cuadrado \* fm\_lado\_cuadrado; // Calculando área del cuadrado  
printf("Área del cuadrado: %.2f\n", fm\_area\_cuadrado);

// Solicitando datos para el rectángulo  
printf("Ingrese la base del rectángulo: ");  
scanf("%f", &fm\_base\_rectangulo);  
printf("Ingrese la altura del rectángulo: ");  
scanf("%f", &fm\_altura\_rectangulo);  
fm\_area\_rectangulo = fm\_base\_rectangulo \* fm\_altura\_rectangulo;   
printf("Área del rectángulo: %.2f\n", fm\_area\_rectangulo);  
  
// Solicitando datos para el triángulo  
printf("Ingrese la base del triángulo: ");  
scanf("%f", &fm\_base\_triangulo);  
printf("Ingrese la altura del triángulo: ");  
scanf("%f", &fm\_altura\_triangulo);  
fm\_area\_triangulo = (fm\_base\_triangulo \* fm\_altura\_triangulo) / 2  
printf("Área del triángulo: %.2f\n", fm\_area\_triangulo);  
  
// Solicitando datos para el círculo  
printf("Ingrese el radio del círculo: ");  
scanf("%f", &fm\_radio\_circulo);  
fm\_area\_circulo = M\_PI \* pow(fm\_radio\_circulo, 2); // Calculando área del círculo  
printf("Área del círculo: %.2f\n", fm\_area\_circulo);  
return 0;

}