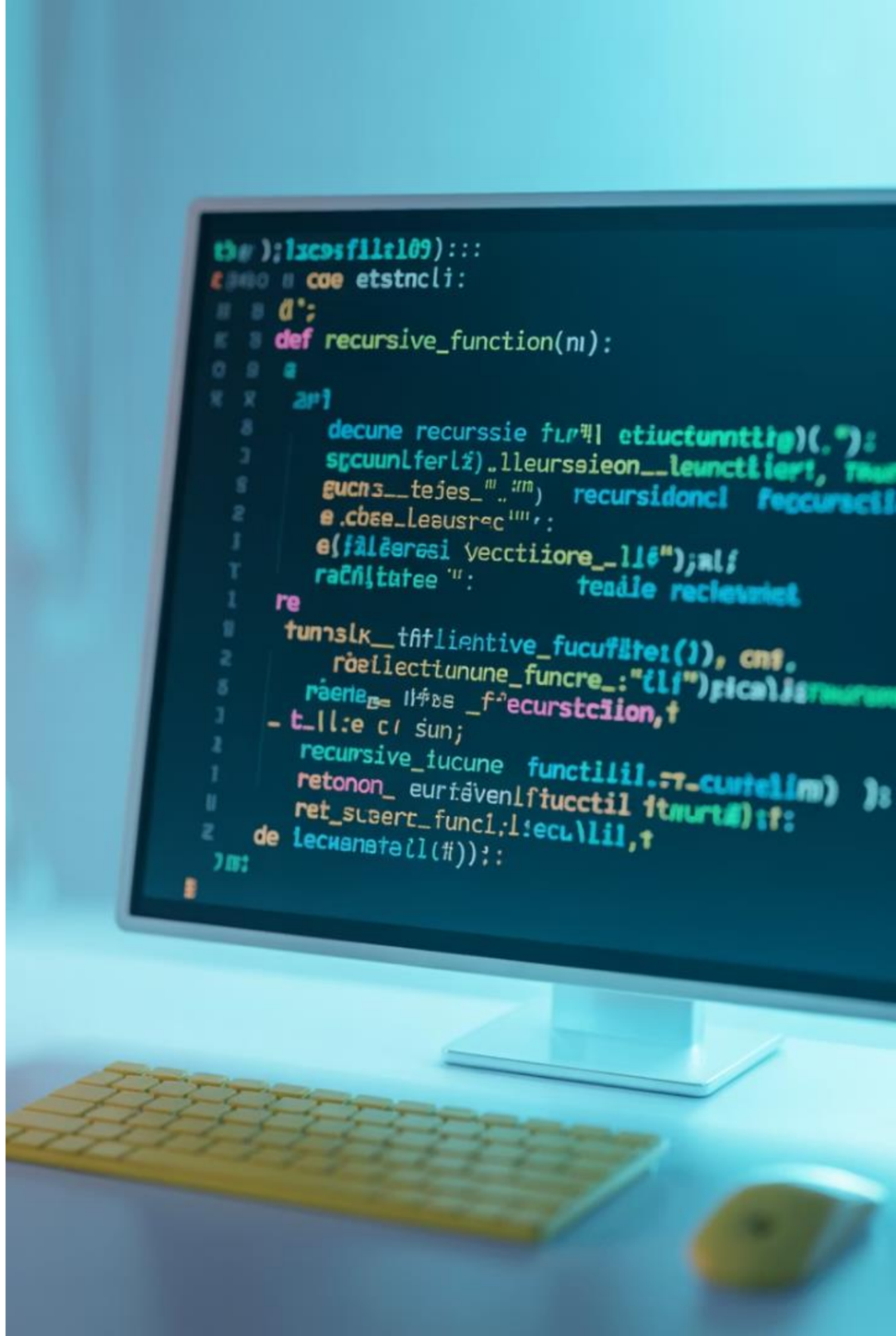
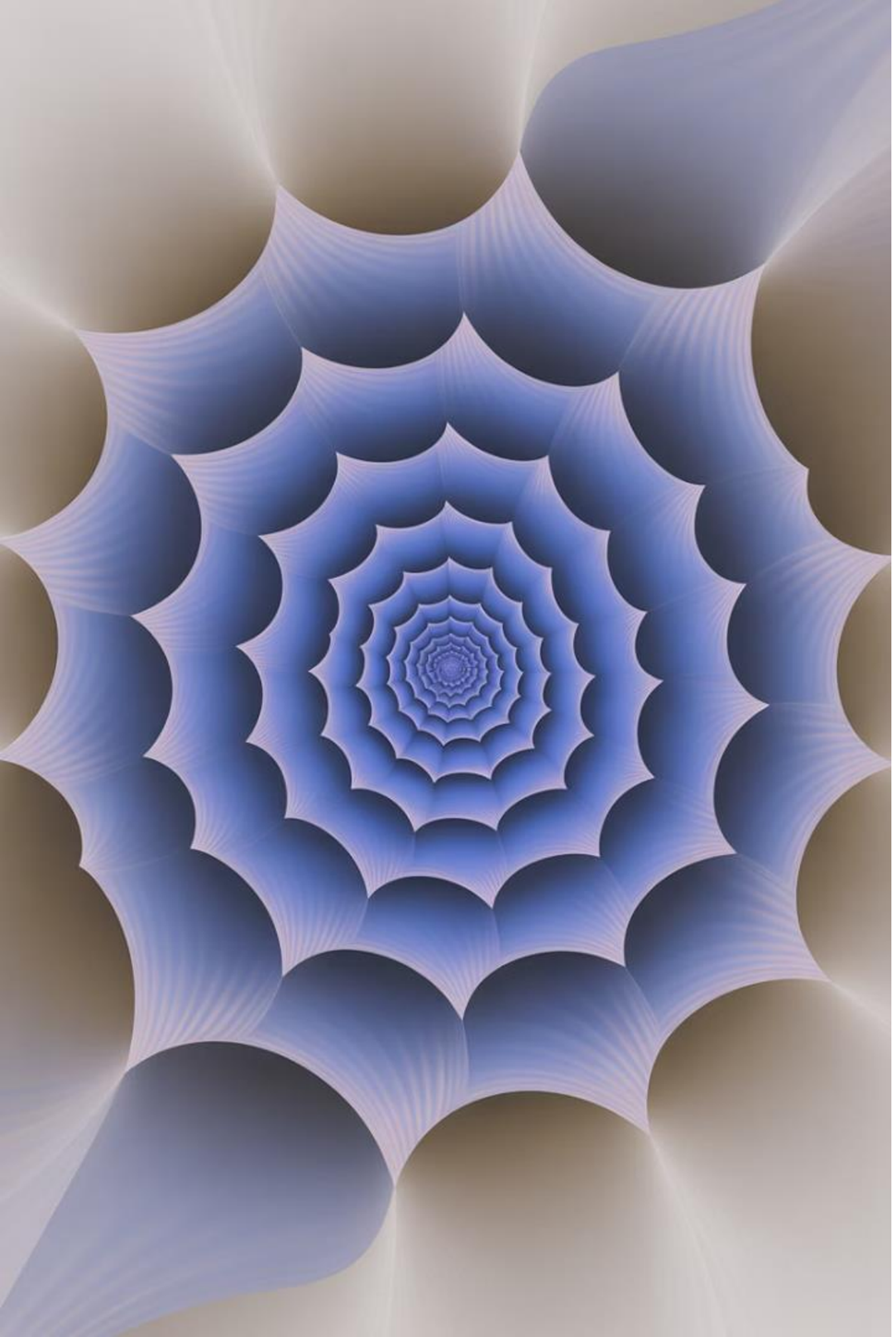
Resolución de ejercicios con recursividad en Python

Exploraremos tres ejercicios prácticos que demuestran el poder de la recursividad: cálculo de potencia, conversión binaria y verificación de palíndromos.





¿Qué es la recursividad?

Concepto fundamental

Técnica donde una función se llama a sí misma para resolver problemas dividiéndolos en subproblemas más pequeños.

Elementos clave

Requiere un caso base (condición de parada) y una parte recursiva que reduzca el problema original.

Aplicación ideal

Se utiliza cuando el problema se puede descomponer en partes similares más pequeñas.

Cálculo de Potencia

</>

Definición de la función

Implementamos la función potencia(base, exponente) con un caso base cuando el exponente es cero.

f(x)

Caso recursivo

Multiplicamos la base por el resultado de potencia(base, exponente-1).

Ejecución del ejemplo

Con base=2 y exponente=3, la función realiza múltiples llamadas anidadas.

=

Resultado final

Obtenemos 2×2×2×1 = 8 al resolverse todas las llamadas recursivas.

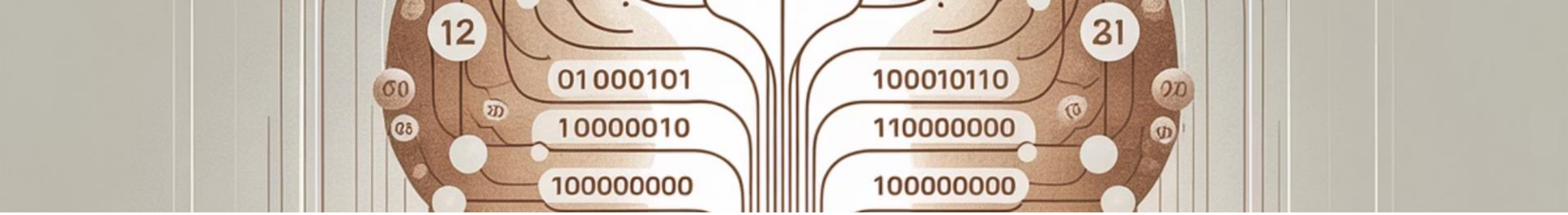
Conversión de Potencia - Código

Implementación recursiva de la función de potencia en Python:

```
def potencia(base, exponente):
    # Caso base: cualquier número elevado a 0 es 1
    if exponente == 0:
        return 1
    # Caso recursivo: base * potencia(base, exponente-1)
    else:
        return base * potencia(base, exponente-1)

# Ejemplo de uso
resultado = potencia(2, 3)
print(f"2^3 = {resultado}")  # Imprime: 2^3 = 8
```

¿Cómo funciona?

- La función verifica primero el **caso base**: si el exponente es 0, retorna 1
- En el **caso recursivo**, multiplica la base por el resultado de llamar a la misma función con el exponente reducido en 1
- Ejemplo con potencia(2,3): se convierte en 2 × potencia(2,2), luego 2 × 2 × potencia(2,1), después 2 × 2 × 2 × potencia(2,0), y finalmente 2 × 2 × 2 × 1 = 8



Conversión de Decimal a Binario

1

Definición del algoritmo

La función decimal_a_binario(n) convierte números decimales a su representación binaria.

2

Caso base

Cuando n=0, devolvemos una cadena vacía como punto de parada.

3

Paso recursivo

Combinamos el resultado de convertir n//2 con el resto n%2.

4

Ejemplo práctico

Al convertir 10, obtenemos "1010" mediante llamadas recursivas sucesivas.

Conversión de Decimal a Binario - Código

Veamos la implementación en Python de nuestra función recursiva para convertir números decimales a binarios:

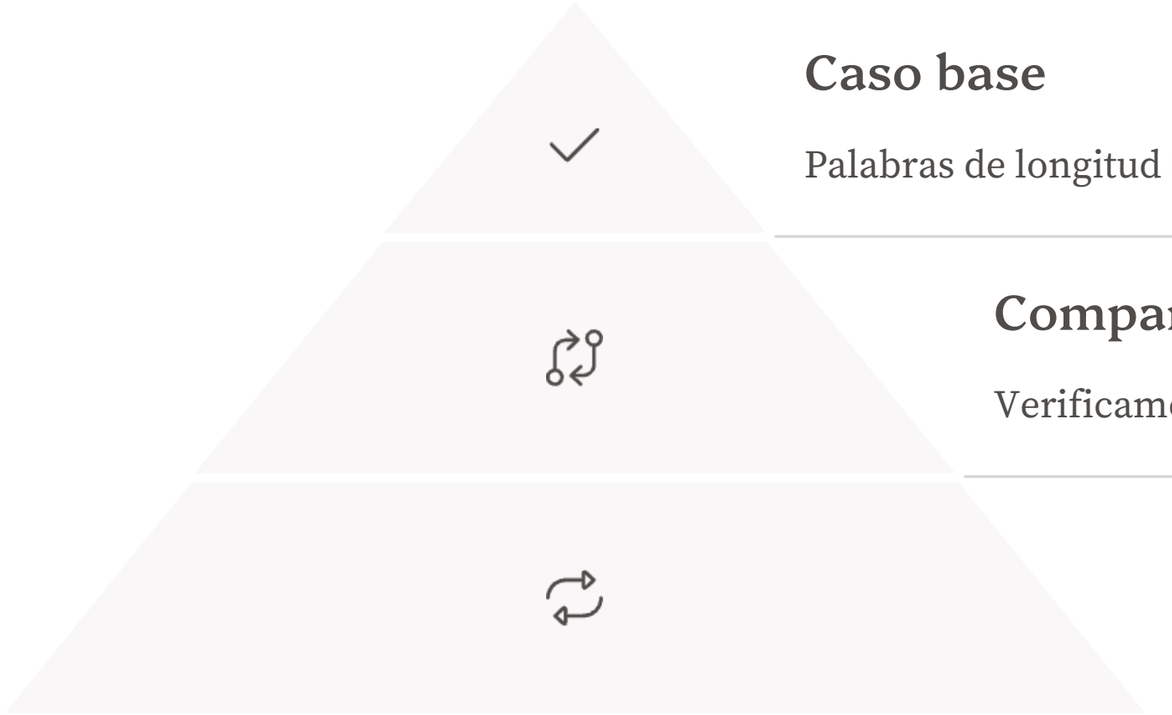
```
def decimal_a_binario(n):
    # Caso base: si el número es 0, devolvemos cadena vacía
    if n == 0:
        return ""
    # Caso recursivo: convertimos n//2 y añadimos el residuo
    else:
        return decimal_a_binario(n // 2) + str(n % 2)

# Ejemplo de uso
numero = 10
print(f"{numero} en binario es: {decimal_a_binario(numero)}")
# Resultado: 10 en binario es: 1010
```

Explicación del funcionamiento:

- La función verifica primero si hemos llegado al caso base (n = 0).
- En cada llamada recursiva, dividimos el número entre 2 (n // 2) y calculamos el residuo (n % 2).
- El residuo (0 o 1) se concatena al final del resultado de la llamada recursiva.
- Las llamadas se anidan hasta llegar a n = 0, y luego se resuelven de adentro hacia afuera.

Esta implementación aprovecha la naturaleza recursiva del algoritmo de conversión, donde cada dígito binario se determina mediante divisiones sucesivas por 2.

Verificación de Palíndromo



Caso base

Palabras de longitud 0 o 1 son palíndromos por definición.

Comparación de extremos

Verificamos si el primer y último carácter son iguales.

Recursión interna

Aplicamos el mismo proceso a la subcadena interior.

Verificación de Palíndromo - Código

La implementación en Python de una función recursiva para verificar palíndromos:

```
def es_palindromo(texto):
    # Caso base: cadenas vacías o de un solo carácter son palíndromos
    if len(texto) <= 1:
        return True

    # Comparamos el primer y último carácter
    if texto[0] != texto[-1]:
        return False

    # Llamada recursiva con la subcadena interior
    return es_palindromo(texto[1:-1])

# Ejemplos de uso
print(es_palindromo("radar"))    # True
print(es_palindromo("python"))   # False
print(es_palindromo("reconocer")) # True
```

Explicación del funcionamiento:

1. La función recibe una cadena de texto como parámetro
2. Si la cadena tiene 0 o 1 caracteres, es un palíndromo por definición (caso base)
3. Comparamos el primer carácter (texto[0]) con el último (texto[-1])
4. Si son diferentes, la palabra no es un palíndromo y devolvemos False
5. Si son iguales, llamamos recursivamente a la función con la subcadena interior (sin el primer y último carácter)
6. El proceso continúa hasta que se reduce a una cadena vacía o de un solo carácter

Esta implementación demuestra cómo la recursividad permite resolver el problema dividiendo la verificación en pasos más pequeños, comparando los extremos y reduciendo gradualmente el tamaño del problema.

Conclusión

Soluciones elegantes

La recursividad proporciona enfoques concisos para problemas complejos.

Caso base crucial

Una condición de parada bien definida evita bucles infinitos.

Aplicación práctica

Los tres ejemplos muestran técnicas aplicables a diversos problemas.

Patrones recursivos

Identificar la estructura repetitiva facilita la implementación.