Explicación detallada de ejercicios sobre recursividad

☑ 3) Potencia recursiva

```
def potencia(base, exponente):
    if exponente == 0:
        return 1
    else:
        return base * potencia(base, exponente - 1)
```

Explicación paso a paso:

La función `potencia` calcula el resultado de elevar una base a un exponente entero positivo utilizando recursividad.
Esto significa que la operación se resuelve en pasos sucesivos, donde cada paso es una llamada a la misma función con un exponente menor.

◇ *Caso base:*

Cuando el exponente es 0, la función devuelve 1, ya que cualquier número elevado a la 0 da 1.
Esto detiene las llamadas recursivas.

◇ *Paso recursivo:*

Se multiplica la `base` por el resultado de la misma función llamada con el `exponente - 1`.

◇ *Ejemplo detallado:*

Queremos calcular 2^3, es decir: `potencia(2, 3)`

```
potencia(2, 3)
→ 2 * potencia(2, 2)
→ 2 * (2 * potencia(2, 1))
→ 2 * (2 * (2 * potencia(2, 0)))
→ 2 * (2 * (2 * 1))
→ 2 * (2 * 2)
→ 2 * 4
→ 8
```

En cada paso, la pila de llamadas se acumula y resuelve desde el fondo hacia arriba.

☑ 4) Conversión de decimal a binario

```
def decimal_a_binario(n):
    if n == 0:
        return ""
    else:
        return decimal_a_binario(n // 2) + str(n % 2)
```

Explicación paso a paso:

Esta función convierte un número decimal (base 10) a binario (base 2) utilizando recursividad.

- *Caso base:*

Cuando el número es 0, se devuelve una cadena vacía. Este es el punto donde se detiene la recursión.

- *Paso recursivo:*

La función divide el número entre 2 (n // 2) y concatena el resto de esa división (n % 2) al final del resultado.

- *Ejemplo detallado:*

Convertimos el número `10` a binario:

```
decimal_a_binario(10)
→ decimal_a_binario(5) + '0'      # 10 % 2 = 0
→ (decimal_a_binario(2) + '1') + '0'      # 5 % 2 = 1
→ ((decimal_a_binario(1) + '0') + '1') + '0'      # 2 % 2 = 0
→ (((decimal_a_binario(0) + '1') + '0') + '1') + '0'      # 1 % 2 = 1
→ "" + '1' + '0' + '1' + '0' = "1010"
```

Es importante destacar que el sistema binario se construye **de atrás hacia adelante**, y por eso la función concatena los restos después de resolver todas las llamadas recursivas.

☑ 5) Verificación de palíndromo

```
def es_palindromo(palabra):
    if len(palabra) <= 1:
        return True
    if palabra[0] != palabra[-1]:
        return False
    return es_palindromo(palabra[1:-1])
```

Explicación paso a paso:

Un **palíndromo** es una palabra que se lee igual hacia adelante y hacia atrás (como "radar", "reconocer", "oso").

◇ *Caso base:*

Si la palabra tiene 0 o 1 letras, se considera un palíndromo (ya que no hay más letras que comparar).

◇ *Paso recursivo:*

La función compara el primer carácter (`palabra[0]`) con el último (`palabra[-1]`).

- Si **son diferentes**, la palabra no es un palíndromo.
- Si **son iguales**, la función se llama a sí misma con la subcadena que queda en el medio (`palabra[1:-1]`), eliminando el primer y último carácter.

◇ *Ejemplo detallado:*

Verificar si `"radar"` es un palíndromo:

```
es_palindromo("radar")
→ 'r' == 'r', sigue con "ada"
→ 'a' == 'a', sigue con "d"
→ longitud = 1 → devuelve True
```

La función va "recortando" la palabra desde los extremos hacia el centro y comprobando si sigue siendo simétrica.