# Ejercicios de Abstracciones funcionales

**Programación — DAW**

Ricardo Pérez López
IES Doñana

Curso 2025/2026

1. Escribir en forma de función todos los programas solicitados en el boletín de ejercicios de «Programación funcional». Para cada una de ellas, dar un ejemplo de uso.

2. Escribir una función que implemente la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{repite(}s\texttt{: str, }n\texttt{: int) -> str} \\ \textbf{Post}: & \texttt{repite(}s\texttt{, }n\texttt{)} = s \text{ repetido } n \text{ veces} \end{cases}$$

   Dar un ejemplo de uso.

3. Escribir una función que implemente la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{len(}c\texttt{)} = 1 \\ & \texttt{es\_vocal(}c\texttt{: str) -> bool} \\ \textbf{Post}: & \texttt{es\_vocal(}c\texttt{)} = c \text{ es una vocal, acentuada o no} \end{cases}$$

   Dar un ejemplo de uso.

4. Escribir una función `calculadora` a la que se le pasan dos números reales y qué operación se desea realizar con ellos. Las operaciones disponibles son: sumar, restar, multiplicar o dividir. Éstas se especifican mediante un carácter: `'+'`, `'-'`, `'*'` o `'/'`, respectivamente. La función devolverá el resultado de la operación en forma de número real.

5. Escribir una función que calcule la distancia euclídea entre dos puntos $(x_1, y_1)$ y $(x_2, y_2)$, descrita por la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{distancia(}x_1\texttt{: float, }y_1\texttt{: float, }x_2\texttt{: float, }y_2\texttt{: float) -> float} \\ \textbf{Post}: & \texttt{distancia(}x_1\texttt{, }y_1\texttt{, }x_2\texttt{, }y_2\texttt{)} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \end{cases}$$

Dar un ejemplo de uso.

6. Escribir una función que reciba una cantidad de días, minutos y segundos, y que calcule y devuelva el número de segundos en los datos de entrada indicados. Dar un ejemplo de uso.

7. Escribir una función que reciba dos instantes de tiempo en forma de horas y minutos y que cumpla con la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & hora_1 \geq 0 \land minuto_1 \geq 0 \land hora_2 \geq 0 \land minuto_2 \geq 0 \\ & \texttt{distancia(}hora_1\texttt{: int, }minuto_1\texttt{: int, }hora_2\texttt{: int, }minuto_2\texttt{: int) -> int} \\ \textbf{Post}: & \texttt{distancia(}hora_1\texttt{, }minuto_1\texttt{, }hora_2\texttt{, }minuto_2\texttt{)} = \\ & \text{la cantidad de minutos que existen de diferencia entre los dos instantes} \end{cases}$$

Dar un ejemplo de uso.

8. Dada la siguiente función matemática:

$$f(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 + 2 \cdot f(n-1) & \text{si } n > 0 \end{cases}$$

calcular el valor de $f(3)$.

9. La función `potencia` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & b \geq 0 \\ & \texttt{potencia(}a\texttt{: int, }b\texttt{: int) -> int} \\ \textbf{Post}: & \texttt{potencia(}a\texttt{, }b\texttt{)} = a^b \end{cases}$$

   *a)* Implementar la función de forma no recursiva.

   *b)* Implementar la función de forma recursiva.

10. La función `repite` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{repite(}s\texttt{: str, }n\texttt{: int) -> str} \\ \textbf{Post}: & \texttt{repite(}s\texttt{, }n\texttt{)} = s \ast n \end{cases}$$

Implementar la función de forma recursiva.

11. La suma lenta es un algoritmo para sumar dos números para el que sólo necesitamos saber cuáles son el anterior y el siguiente de un número dado. El algoritmo se basa en la siguiente recurrencia:

$$suma\_lenta(a, b) = \begin{cases} b & \text{si } a = 0 \\ suma\_lenta(ant(a), sig(b)) & \text{si } a > 0 \end{cases}$$

Suponiendo que tenemos las siguientes funciones `ant` y `sig`:

```
ant = lambda n: n - 1
sig = lambda n: n + 1
```

Se pide:

*a)* Escribir su especificación.

*b)* Implementar una función recursiva que satisfaga dicha especificación.

12. La función `suma_digitos` calcula la suma de los dígitos de un número entero:

```
suma_digitos(423) = 4 + 2 + 3 = 9
suma_digitos(7) = 7
```

Se pide:

*a)* Escribir su especificación.

*b)* Implementar una función recursiva que satisfaga dicha especificación.

*Indicación*: Recordar que `n // 10` le quita el último dígito a $n$. Además, `n % 10` devuelve el último dígito de $n$.

13. La función `voltea` le da la vuelta a un número entero:

```
voltea(423) = 324
voltea(7) = 7
```

Se pide:

*a)* Escribir su especificación.

*b)* Implementar una función recursiva que satisfaga dicha especificación.

*Indicación*: Usar la función `digitos` que devuelve la cantidad de dígitos que tiene un entero. Usar además la indicación del ejercicio anterior.

14. La función `par_positivo` determina si un número entero positivo es par:

```
par_positivo(0) = True
par_positivo(1) = False
par_positivo(27) = False
par_positivo(82) = True
```

Se pide:

*a)* Escribir su especificación.

*b)* Implementar una función recursiva que satisfaga dicha especificación.

15. La función `par` determina si un número entero (positivo o negativo) es par:

```
par(0) = True
par(1) = False
par(-27) = False
```

Se pide:

*a)* Escribir su especificación.

*b)* Implementar una función recursiva que satisfaga dicha especificación.

*c)* ¿Cómo se podría implementar una función `impar` a partir de la función `par`?

16. La función `elem` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{elem(}e\texttt{, }t\texttt{: tuple) -> bool} \\ \textbf{Post}: & \texttt{elem(}e\texttt{, }t\texttt{)} = \begin{cases} \texttt{True} & \text{si } e \text{ está en } t \\ \texttt{False} & \text{en caso contrario} \end{cases} \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación.

17. La función `cuantos` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{cuantos(}e\texttt{, }t\texttt{: tuple) -> int} \\ \textbf{Post}: & \texttt{cuantos(}e\texttt{, }t\texttt{)} = \text{el número de veces que aparece } e \text{ en } t \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación y que genere un proceso:

*a)* recursivo.

*b)* iterativo.

18. La función `quita` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{quita(}e\texttt{, }t\texttt{: tuple) -> tuple} \\ \textbf{Post}: & \texttt{quita(}e\texttt{, }t\texttt{)} = \text{una tupla igual que } t \text{ pero sin los } e \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación y que genere un proceso:

*a)* recursivo.

*b)* iterativo.

19. La función `sustituye` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{sustituye(}a\texttt{, }b\texttt{, }t\texttt{: tuple) -> tuple} \\ \textbf{Post}: & \texttt{sustituye(}a\texttt{, }b\texttt{, }t\texttt{)} = \text{una tupla igual que } t \text{ pero} \\ & \text{sustituyendo los } a \text{ por } b \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación y que genere un proceso:

*a)* recursivo.

*b)* iterativo.

20. La función `ultimo` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \\ & \texttt{ultimo(}t\texttt{: tuple) -> Any} \\ \textbf{Post}: & \texttt{ultimo(}t\texttt{)} = \text{el último elemento de } t \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación.

21. La función `enesimo` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \land 0 \leq n < \texttt{len(}t\texttt{)} \\ & \texttt{enesimo(}n\texttt{: int, }t\texttt{: tuple) -> Any} \\ \textbf{Post}: & \texttt{enesimo(}n\texttt{, }t\texttt{)} = \text{el } n\text{-ésimo elemento de } t \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación.

22. La función `invertir` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{invertir(}t\texttt{: tuple) -> tuple} \\ \textbf{Post}: & \texttt{invertir(}t\texttt{)} = \text{una tupla con los elementos de } t \text{ en orden contrario} \end{cases}$$

Por ejemplo: `invertir((1, 2, 3, 4)) == (4, 3, 2, 1)`.

Escribir una función recursiva que satisfaga dicha especificación.

23. La función `palindromo` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{palindromo(}t\texttt{: tuple) -> bool} \\ \textbf{Post}: & \texttt{palindromo(}t\texttt{)} = \begin{cases} \texttt{True} & \text{si } t \text{ es un palíndromo} \\ & \text{(se lee igual en un sentido que en otro)} \\ \texttt{False} & \text{en caso contrario} \end{cases} \end{cases}$$

Por ejemplo: `palindromo((1, 2, 3, 4, 3, 2, 1)) == True`.

Escribir una función recursiva que satisfaga dicha especificación.

24. La función `rota` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{rota}(n\texttt{: int, } t\texttt{: tuple) -> tuple} \\ \textbf{Post}: & \texttt{rota}(n\texttt{, } t\texttt{)} = \text{la tupla obtenida poniendo los } n \text{ primeros} \\ & \text{elementos de } t \text{ al final} \end{cases}$$

Por ejemplo:

```
rota(1, (3, 2, 5, 7)) == (2, 5, 7, 3)
rota(2, (3, 2, 5, 7)) == (5, 7, 3, 2)
rota(3, (3, 2, 5, 7)) == (7, 3, 2, 5)
```

Escribir una función recursiva que satisfaga dicha especificación.

25. La función `rota1` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{rota1(}t\texttt{: tuple) -> tuple} \\ \textbf{Post}: & \texttt{rota1(}t\texttt{)} = \text{la tupla obtenida poniendo el primer} \\ & \text{elemento de } t \text{ al final} \end{cases}$$

Por ejemplo: `rota1((3, 2, 5, 7)) == (2, 5, 7, 3)`.

Escribir una función recursiva que satisfaga dicha especificación.

26. La función `menor` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \\ & \texttt{menor(}t\texttt{: tuple[}\alpha\texttt{]) -> } \alpha \\ \textbf{Post}: & \texttt{menor(}t\texttt{)} = \text{el menor elemento de } t \end{cases}$$

Por ejemplo: `menor((3, 2, 5, 7)) == 2`.

Escribir una función recursiva que satisfaga dicha especificación.

27. La función `mayor` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \\ & \texttt{mayor(}t\texttt{: tuple[}\alpha\texttt{]) -> } \alpha \\ \textbf{Post}: & \texttt{mayor(}t\texttt{)} = \text{el mayor elemento de } t \end{cases}$$

Por ejemplo: `mayor((3, 2, 5, 7)) == 7`.

Escribir una función recursiva que satisfaga dicha especificación.

28. La función `menor_mayor` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \\ & \texttt{menor\_mayor}(t\texttt{: tuple[}\alpha\texttt{]) -> tuple[}\alpha\texttt{]} \\ \textbf{Post}: & \texttt{menor\_mayor}(t) = \text{una tupla con dos elementos} \\ & \text{que contiene el menor y el mayor elemento de } t, \\ & \text{en ese orden} \end{cases}$$

Por ejemplo: `menor_mayor((3, 2, 5, 7)) == (2, 7)`.

Escribir una función recursiva que satisfaga dicha especificación.

29. La función `finales` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{finales}(n\texttt{: int, }t\texttt{: tuple) -> tuple} \\ \textbf{Post}: & \texttt{finales}(n, t) = \text{la tupla que contiene los } n \text{ elementos finales de } t \end{cases}$$

Por ejemplo: `finales(2, (1, 2, 3, 4)) == (3, 4)`.

Escribir una función recursiva que satisfaga dicha especificación.

# Soluciones

1. $f(3)$
   $= 1 + 2 \cdot f(2)$
   $= 1 + 2 \cdot (1 + f(1))$
   $= 1 + 2 \cdot (1 + 2 \cdot (1 + 2 \cdot f(0)))$
   $= 1 + 2 \cdot (1 + 2 \cdot (1 + 2 \cdot 0))$
   $= 1 + 2 \cdot (1 + 2 \cdot 1)$
   $= 1 + 2 \cdot 3$
   $= 7.$

2. *a)* `potencia = lambda a, b: a ** b`

   *b)* `potencia = lambda a, b: 1 if b == 0 else a * potencia(a, b - 1)`

3. `repite = lambda s, n: '' if n == 0 else s + repite(s, n - 1)`

4. *a)*
$$\begin{cases} \textbf{Pre}: & a \geq 0 \\ & \texttt{suma\_lenta}(a\texttt{: int, }b\texttt{: int) -> int} \\ \textbf{Post}: & \texttt{suma\_lenta}(a, b) = a + b \end{cases}$$

   *b)* `suma_lenta = lambda a, b: b if a == 0 else suma_lenta(ant(a), sig(b))`

5. *a)*
$$\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{suma\_digitos}(n\texttt{: int) -> int} \\ \textbf{Post}: & \texttt{suma\_digitos}(n) = \text{la suma de los dígitos de } n \end{cases}$$

   *b)* `suma_digitos = lambda n: n if n < 10 else (n % 10) + suma_digitos(n // 10)`

6. *a)*
$$\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{voltea}(n\texttt{: int) -> int} \\ \textbf{Post}: & \texttt{voltea}(n) = \text{el número } n \text{ con los dígitos al revés} \end{cases}$$

   *b)*
```
voltea = lambda n: n if n < 10 else \
                   (n % 10) * 10 ** (digitos(n) - 1) + voltea(n // 10)
```

7.

*a)* $\begin{cases} \textbf{Pre}: & n \geq 0 \\ & \texttt{par\_positivo}(n\texttt{: int}) \texttt{ -> bool} \\ \textbf{Post}: & \texttt{par\_positivo}(n) = \begin{cases} \texttt{True} & \text{si } n \text{ es par} \\ \texttt{False} & \text{en caso contrario} \end{cases} \end{cases}$

*b)*
```
par_positivo = lambda n: True if n == 0 else \
                             not par_positivo(n - 1)
```

8. *a)* $\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{par(n: int) -> bool} \\ \textbf{Post}: & \texttt{par}(n) = \begin{cases} \texttt{True} & \text{si } n \text{ es par} \\ \texttt{False} & \text{en caso contrario} \end{cases} \end{cases}$

   *b)*
```
par = lambda n: True if n == 0 else \
                not par(abs(n) - 1)
```

   *c)*
```
impar = lambda n: not par(n)
```

9.
```
elem = lambda e, t: False if t == () else \
                    True if t[0] == e else \
                    elem(e, t[1:])
```

10. Definimos:

```
aux = lambda a, b: 1 if a == b else 0
```

   *a)*
```
cuantos = lambda e, t: 0 if t == () else \
                       aux(e, t[0]) + cuantos(e, t[1:])
```

   *b)*
```
cuantos = lambda e, t: cuantos_it(e, t, 0)
cuantos_it = lambda e, t, acc: acc if t == () else \
                               cuantos_it(e, t[1:], acc + aux(e, t[0]))
```

11. Definimos:

```
aux = lambda a, b: () if a == b else (b,)
```

   *a)*
```
quita = lambda e, t: () if t == () else \
                     aux(e, t[0]) + quita(e, t[1:])
```

   *b)*
```
quita = lambda e, t: quita_it(e, t, ())
quita_it = lambda e, t, acc: acc if t == () else \
                             quita_it(e, t[1:], acc + aux(e, t[0]))
```

12. Definimos:

```
aux = lambda a, b, t: (b,) if a == t else (t,)
```

*a)*
```
sustituye = lambda a, b, t: () if t == () else \
                            aux(a, b, t[0]) + sustituye(a, b, t[1:])
```

*b)*
```
sustituye = lambda a, b, t: sustituye_it(a, b, t, ())
sustituye_it = lambda a, b, t, acc: \
    acc if t == () else \
    sustituye_it(a, b, t[1:], acc + aux(a, b, t[0]))
```

13.
```
ultimo = lambda t: t[0] if t[1:] == () else ultimo(t[1:])
```

14.
```
enesimo = lambda n, t: t[0] if n == 0 else enesimo(n - 1, t[1:])
```