

# Ejercicios de Programación funcional (II)

Programación — DAW

Ricardo Pérez López  
IES Doñana

Curso 2020/2021

1. 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  $f(3)$ .

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

$$\left\{ \begin{array}{l} \text{Pre : } b \geq 0 \\ \text{potencia}(a : \text{int}, b : \text{int}) \rightarrow \text{int} \\ \text{Post : } \text{potencia}(a, b) = a^b \end{array} \right.$$

- a) Implementar la función de forma no recursiva.  
b) Implementar la función de forma recursiva.
3. La función `repite` tiene la siguiente especificación:

$$\left\{ \begin{array}{l} \text{Pre : } n \geq 0 \\ \text{repite}(s : \text{str}, n : \text{int}) \rightarrow \text{str} \\ \text{Post : } \text{repite}(s, n) = s * n \end{array} \right.$$

Implementar la función de forma recursiva.

4. 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.
5. 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`.

6. 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.

7. 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:

- Escribir su especificación.
  - Implementar una función recursiva que satisfaga dicha especificación.
8. La función `par` determina si un número entero (positivo o negativo) es par:

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

Se pide:

- Escribir su especificación.
  - Implementar una función recursiva que satisfaga dicha especificación.
  - ¿Cómo se podría implementar una función `impar` a partir de la función `par`?
9. La función `elem` tiene la siguiente especificación:

$$\left\{ \begin{array}{l} \text{Pre : } \text{True} \\ \text{elem}(e, t : \text{tuple}) \rightarrow \text{bool} \\ \text{Post : } \text{elem}(e, t) = \begin{cases} \text{True} & \text{si } e \text{ está en } t \\ \text{False} & \text{en caso contrario} \end{cases} \end{array} \right.$$

Escribir una función recursiva que satisfaga dicha especificación.

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

$$\left\{ \begin{array}{l} \text{Pre : } \text{True} \\ \text{cuantos}(e, t : \text{tuple}) \rightarrow \text{int} \\ \text{Post : } \text{cuantos}(e, t) = \text{el número de veces que aparece } e \text{ en } t \end{array} \right.$$

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

- a) recursivo.
- b) iterativo.

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

$$\left\{ \begin{array}{l} \text{Pre : } \text{True} \\ \text{quita}(e, t : \text{tuple}) \rightarrow \text{tuple} \\ \text{Post : } \text{quita}(e, t) = \text{una tupla igual que } t \text{ pero sin los } e \end{array} \right.$$

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

- a) recursivo.
- b) iterativo.

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

$$\left\{ \begin{array}{l} \text{Pre : } \text{True} \\ \text{sustituye}(a, b, t : \text{tuple}) \rightarrow \text{tuple} \\ \text{Post : } \text{sustituye}(a, b, t) = \text{una tupla igual que } t \text{ pero} \\ \text{sustituyendo los } a \text{ por } b \end{array} \right.$$

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

- a) recursivo.
- b) iterativo.

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

$$\left\{ \begin{array}{l} \text{Pre : } t \neq () \\ \text{ultimo}(t : \text{tuple}) \\ \text{Post : } \text{ultimo}(t) = \text{el último elemento de } t \end{array} \right.$$

Escribir una función recursiva que satisfaga dicha especificación.

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

$$\left\{ \begin{array}{l} \textbf{Pre} : \quad t \neq () \wedge 0 \leq n < \text{len}(t) \\ \quad \text{enesimo}(n : \text{int}, t : \text{tuple}) \\ \textbf{Post} : \quad \text{enesimo}(n, t) = \text{el } n\text{-ésimo elemento de } t \end{array} \right.$$

Escribir una función recursiva que satisfaga dicha especificación.

## Soluciones

$$\begin{aligned}
 1. \quad & f(3) \\
 &= 1 + 2 \cdot f(2) \\
 &= 1 + 2 \cdot (1 + f(1)) \\
 &= 1 + 2 \cdot (1 + 2 \cdot (1 + 2 \cdot f(0))) \\
 &= 1 + 2 \cdot (1 + 2 \cdot (1 + 2 \cdot 0)) \\
 &= 1 + 2 \cdot (1 + 2 \cdot 1) \\
 &= 1 + 2 \cdot 3 \\
 &= 7.
 \end{aligned}$$

$$2. \quad a) \text{ potencia} = \text{lambda } a, b: a ** b$$

$$b) \text{ potencia} = \text{lambda } a, b: 1 \text{ if } b == 0 \text{ else } a * \text{potencia}(a, b - 1)$$

$$3. \text{ repite} = \text{lambda } s, n: '' \text{ if } n == 0 \text{ else } s + \text{repite}(s, n - 1)$$

$$4. \quad a) \left\{ \begin{array}{l} \text{Pre: } a \geq 0 \\ \text{suma\_lenta}(a: \text{int}, b: \text{int}) \rightarrow \text{int} \\ \text{Post: } \text{suma\_lenta}(a, b) = a + b \end{array} \right.$$

$$b) \text{ suma\_lenta} = \text{lambda } a, b: b \text{ if } a == 0 \text{ else } \text{suma\_lenta}(\text{ant}(a), \text{sig}(b))$$

$$5. \quad a) \left\{ \begin{array}{l} \text{Pre: } n \geq 0 \\ \text{suma\_digitos}(n: \text{int}) \rightarrow \text{int} \\ \text{Post: } \text{suma\_digitos}(n) = \text{la suma de los dígitos de } n \end{array} \right.$$

$$b) \text{ suma\_digitos} = \text{lambda } n: n \text{ if } n < 10 \text{ else } (n \% 10) + \text{suma\_digitos}(n // 10)$$

$$6. \quad a) \left\{ \begin{array}{l} \text{Pre: } n \geq 0 \\ \text{voltea}(n: \text{int}) \rightarrow \text{int} \\ \text{Post: } \text{voltea}(n) = \text{el número } n \text{ con los dígitos al revés} \end{array} \right.$$

$$b) \text{ voltea} = \text{lambda } n: n \text{ if } n < 10 \text{ else } \backslash \\ (n \% 10) * 10 ** (\text{digitos}(n) - 1) + \text{voltea}(n // 10)$$

$$7. \quad a) \left\{ \begin{array}{l} \text{Pre: } n \geq 0 \\ \text{par\_positivo}(n: \text{int}) \rightarrow \text{bool} \\ \text{Post: } \text{par\_positivo}(n) = \begin{cases} \text{True} & \text{si } n \text{ es par} \\ \text{False} & \text{en caso contrario} \end{cases} \end{array} \right.$$

b) `par_positivo = lambda n: True if n == 0 else \`  
`False if par_positivo(n - 1) else \`  
`True`

8. a) 
$$\left\{ \begin{array}{l} \text{Pre: } \text{True} \\ \text{par}(n: \text{int}) \rightarrow \text{bool} \\ \text{Post: } \text{par}(n) = \begin{cases} \text{True} & \text{si } n \text{ es par} \\ \text{False} & \text{en caso contrario} \end{cases} \end{array} \right.$$

b) `par = lambda n: True if n == 0 else \`  
`False if par(abs(n) - 1) else \`  
`True`

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:])

```