Lenguajes, Tecnologías y Paradigmas de la programación (LTP)

Práctica 4: Programación Funcional

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Sergio Pérez
serperu@dsic.upv.es

Introducción al lenguaje Haskell

OBJETIVOS DE LA PRÁCTICA

- Entender y definir programas Haskell sencillos
- Familiarizarse con el entorno GHCi

El entorno GHCi

```
serperu:Practica4 serperu$ ghci
GHCi, version 8.8.1: https://www.haskell.org/ghc/  :? for help
Prelude>
```

Comandos Básicos

- `:load <module>` (`:l <module>`)
- `:reload` (`:r`)
- `:type <expression>` (`:t <expression>`)
- `:quit` (`:q`)
- `:info <function>` (`:i <function>`)

Notación de operadores en Haskell

NOTACIÓN INFIJA

NOTACIÓN PREFIJA

Notación de operadores en Haskell

NOTACIÓN INFIJA

4 2

NOTACIÓN PREFIJA

Notación de operadores en Haskell

NOTACIÓN INFIJA

4 + 2

NOTACIÓN PREFIJA

Notación de operadores en Haskell

NOTACIÓN INFIJA

4 + 2

NOTACIÓN PREFIJA

4 2

Notación de operadores en Haskell

NOTACIÓN INFIJA

```
4 + 2
```

NOTACIÓN PREFIJA

```
div 4  2
```

Notación de operadores en Haskell

NOTACIÓN INFIJA

4 + 2

Operadores infijos:

+ - / * …

NOTACIÓN PREFIJA

div 4 2

Operadores prefijos:

div mod …

Notación de operadores en Haskell

NOTACIÓN INFIJA

4 + 2

Operadores infijos:

+ - / * …

De infijo a prefijo:

4 `2`

NOTACIÓN PREFIJA

div 4 2

Operadores prefijos:

div mod …

De prefijo a infijo:

4 `2`

Notación de operadores en Haskell

NOTACIÓN INFIJA

4 + 2

Operadores infijos:

+ - / * …

De infijo a prefijo:

(+) 4 2

NOTACIÓN PREFIJA

div 4 2

Operadores prefijos:

div mod …

De prefijo a infijo:

4 `+` 2

Notación de operadores en Haskell

NOTACIÓN INFIJA

```
4 + 2
```

Operadores infijos:

```
+ - / * …
```

De infijo a prefijo:

```
(+) 4  2
```

NOTACIÓN PREFIJA

```
div 4  2
```

Operadores prefijos:

```
div  mod …
```

De prefijo a infijo:

```
4 `div` 2
```

El lenguaje Haskell

- Definir un módulo:

```
module Name where ...
```

NOTA: El nombre del fichero debe ser el mismo que el del módulo y debe tener la extensión "**.hs**". La primera letra del nombre del módulo debe ser **MAYÚSCULA**

```
module Practica4 where
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)

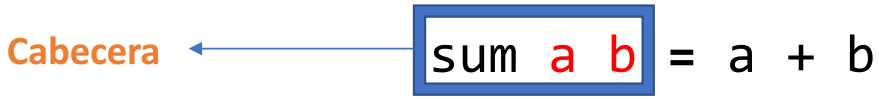
```
sum a b = a + b
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)

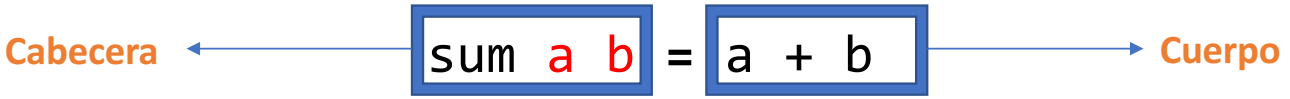


El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)

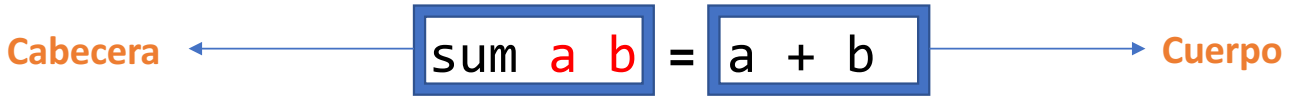


El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)



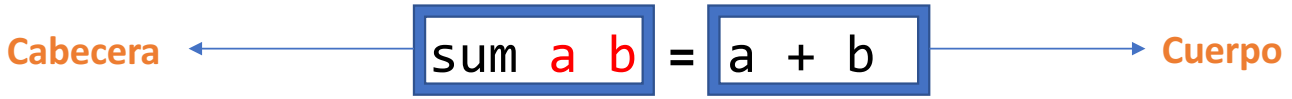

```
¿sum 1 2?
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)



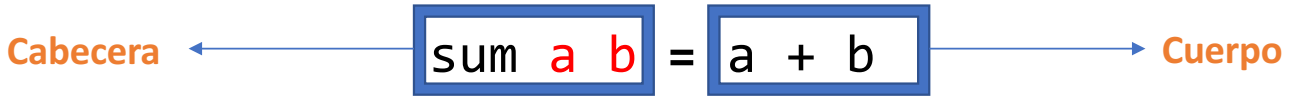

```
¿sum 1 2? => a = 1, b = 2
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)



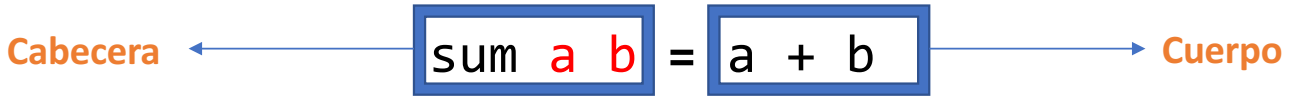

```
¿sum 1 2? => a = 1, b = 2 => 1 + 2
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)



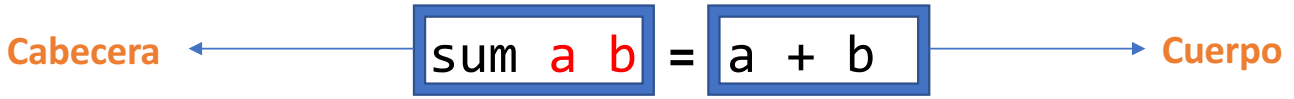

```
¿sum 1 2? => a = 1, b = 2 => 1 + 2 => 3
```

El lenguaje Haskell

- Definir una función:

```
nameFunction arg1 arg2 = ···
```

Ejemplo1: Función suma (`sum`)




```
¿sum 1 2? => a = 1, b = 2 => 1 + 2 => 3
```

NUNCA puedo usar en la parte derecha del "=" una variable que no este definido en la parte izquierda

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)

```
mult 0 b = 0
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)

```
mult 0 b = 0
mult a 0 = 0
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

El lenguaje Haskell

- Definir una función:

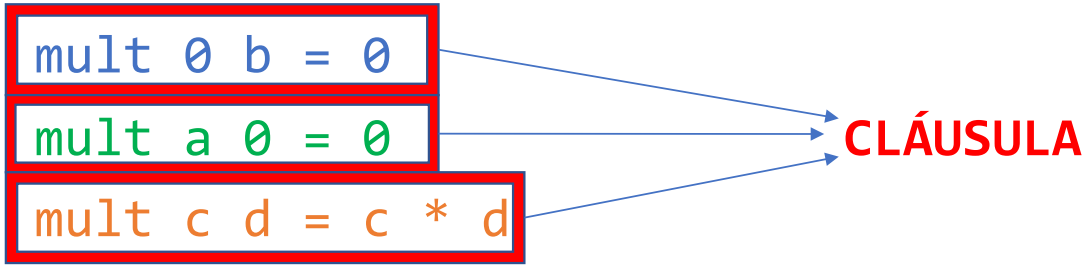
Ejemplo2: Función multiplicar (`mult`)

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

El lenguaje Haskell

- Definir una función:

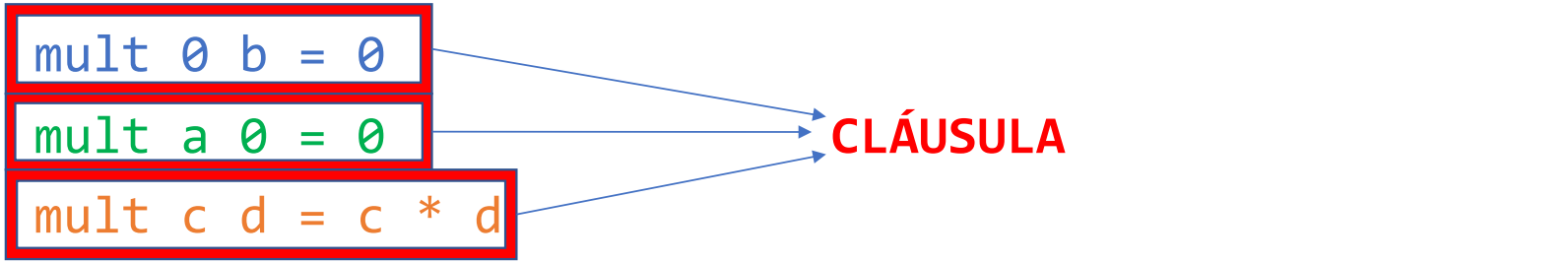
Ejemplo2: Función multiplicar (`mult`)



El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)

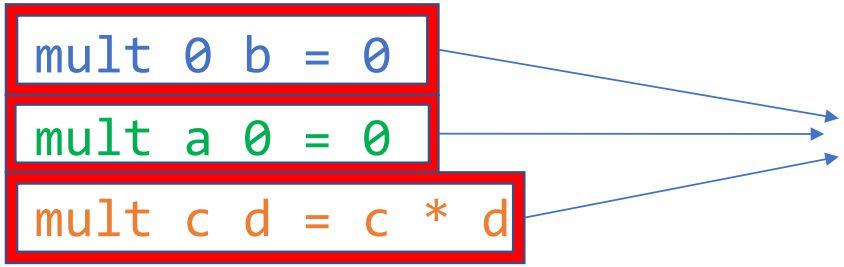


Las Cláusulas son independientes

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)



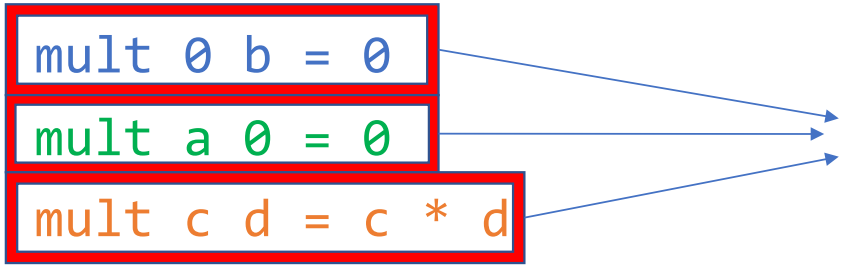
Las Cláusulas son independientes

```
¿mult 2 0? =>
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)



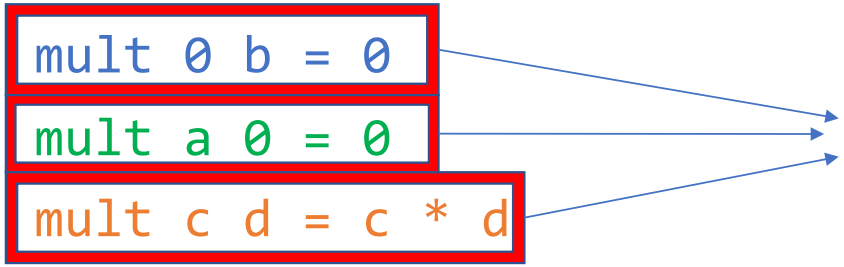
Las Cláusulas son independientes

```
¿mult 2 0? => mult 2 0 => a = 0
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)



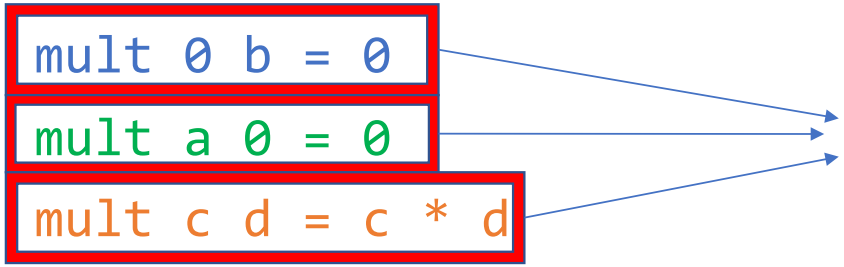
Las Cláusulas son independientes

```
¿mult 2 0? => mult 2 0 => a = 0
¿mult 0 5? =>
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)



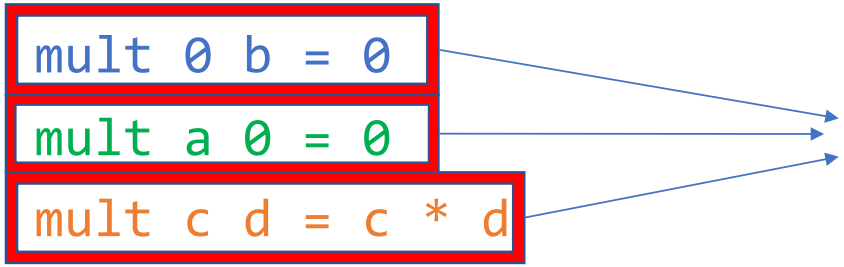
Las Cláusulas son independientes

```
¿mult 2 0? => mult 2 0 => a = 0
¿mult 0 5? => mult 0 5 => b = 5
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)



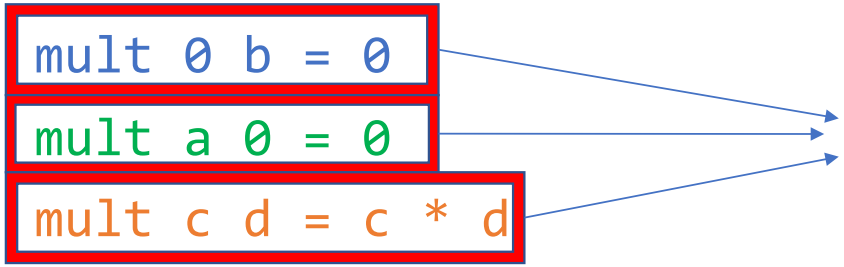
Las Cláusulas son independientes

```
¿mult 2 0? => mult 2 0 => a = 0
¿mult 0 5? => mult 0 5 => b = 5
¿mult 4 8? =>
```

El lenguaje Haskell

- Definir una función:

Ejemplo2: Función multiplicar (`mult`)



Las Cláusulas son independientes

```
¿mult 2 0? => mult 2 0 => a = 0
¿mult 0 5? => mult 0 5 => b = 5
¿mult 4 8? => mult 4 8 => c = 4, d = 8
```

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo a b = ...
```

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

- Pattern matching:

```
foo :: Int -> Int
foo 0 = 1
foo a = a
```

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

- Pattern matching:

```
foo :: Int -> Int
foo 0 = 1
foo a = a
```

foo 5?

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

- Pattern matching:

```
foo :: Int -> Int
foo 0 = 1
foo a = a
```

foo 5?

1) Pattern matching con `foo 0`?

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

- Pattern matching:

```
foo :: Int -> Int
foo 0 = 1
foo a = a
```

`foo 5?`

1) Pattern matching con `foo 0`?

X "5 =/= 0"

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

- Pattern matching:

```
foo :: Int -> Int
foo 0 = 1
foo a = a
```

`foo 5?`

1) Pattern matching con `foo 0`?
X "5 =/= 0"

2) Pattern matching con `foo a`?

El lenguaje Haskell

- El tipo de las funciones:
  - Funcion foo que recibe 2 enteros y devuelve otro entero:

```
foo :: Int -> Int -> Int
foo a b = ...
```

- Pattern matching:

```
foo :: Int -> Int
foo 0 = 1
foo a = a
```

`foo 5?`

1) Pattern matching con `foo 0`?
X "5 =/= 0"

2) Pattern matching con `foo a`?
V asigno a "a" el valor "5"

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

```
mult a b
```

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

```
mult a b
  | a == 0 = 0
```

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

```
mult a b
  | a == 0 = 0
  | b == 0 = 0
```

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

```
mult a b
  | a == 0 = 0
  | b == 0 = 0
  | otherwise = a * b
```

El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

```
mult a b
  | a == 0    = 0
  | b == 0    = 0
  | otherwise = a * b
```

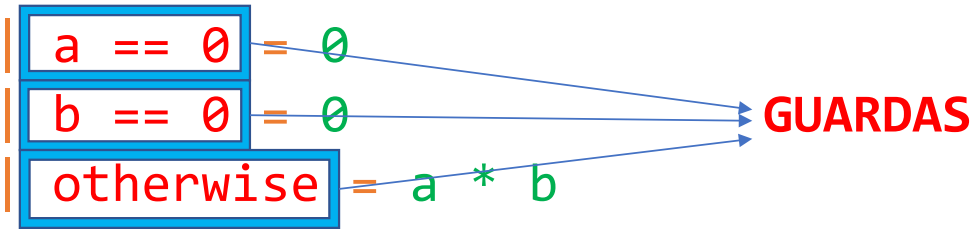
El lenguaje Haskell

- ¿Múltiples cláusulas o guardas?
  - Múltiples cláusulas:

```
mult 0 b = 0
mult a 0 = 0
mult c d = c * d
```

  - Guardas: Sintaxis: | <condición> = <expresión>

```
mult a b
  | a == 0    = 0
  | b == 0    = 0
  | otherwise = a * b
```



Peculiaridades de Haskell

- Formateo del texto (indentación/sangrado)

Es importante que aquellos elementos del mismo tipo tengan el mismo sangrado:

```
module Mymodule where
```

Peculiaridades de Haskell

- Formateo del texto (indentación/sangrado)

Es importante que aquellos elementos del mismo tipo tengan el mismo sangrado:

```
module Mymodule where
··import …
```

Peculiaridades de Haskell

- Formateo del texto (indentación/sangrado)

Es importante que aquellos elementos del mismo tipo tengan el mismo sangrado:

```
module Mymodule where
··import …

··myFunction :: Int -> Int -> Int
··myFunction a b
```

Peculiaridades de Haskell

- Formateo del texto (indentación/sangrado)

Es importante que aquellos elementos del mismo tipo tengan el mismo sangrado:

```
module Mymodule where
··import …

··myFunction :: Int -> Int -> Int
··myFunction a b
····| guard1 = …
····| guard2 = …
```

Peculiaridades de Haskell

- Formateo del texto (indentación/sangrado)

Es importante que aquellos elementos del mismo tipo tengan el mismo sangrado:

```
module Mymodule where
··import …

··myFunction :: Int -> Int -> Int
··myFunction a b
····| guard1 = …
····| guard2 = …
```

```
   Tab character found here, and in one further location.
   Please use spaces instead.
   |
41 |         | a < b = a
   |   ^^^^^^
```

El módulo Data.Char

- Funciones que aparecen en la práctica

```
ord :: Char -> Int
```

`ord a` ➜ *Devuelve el código ASCII de un Char dado*

```
chr :: Int -> Char
```

`chr a` ➜ *Convierte el código ASCII dado (Int) en el Char correspondiente*

El módulo Data.Char

- Funciones que aparecen en la práctica

```
ord :: Char -> Int
```

`ord a` ➜ *Devuelve el código ASCII de un Char dado*

```
chr :: Int -> Char
```

`chr a` ➜ *Convierte el código ASCII dado (Int) en el Char correspondiente*

```
import Data.Char
```

Ejercicios

- Orden recomendado de los ejercicios
  - 3 (con/sin guardas)
  - 6 (recursivo)
  - 1 (importando Char)
  - 2
  - 4 + 5
  - 7

IMPORTANTE: Define el tipo de las funciones en cada ejercicio