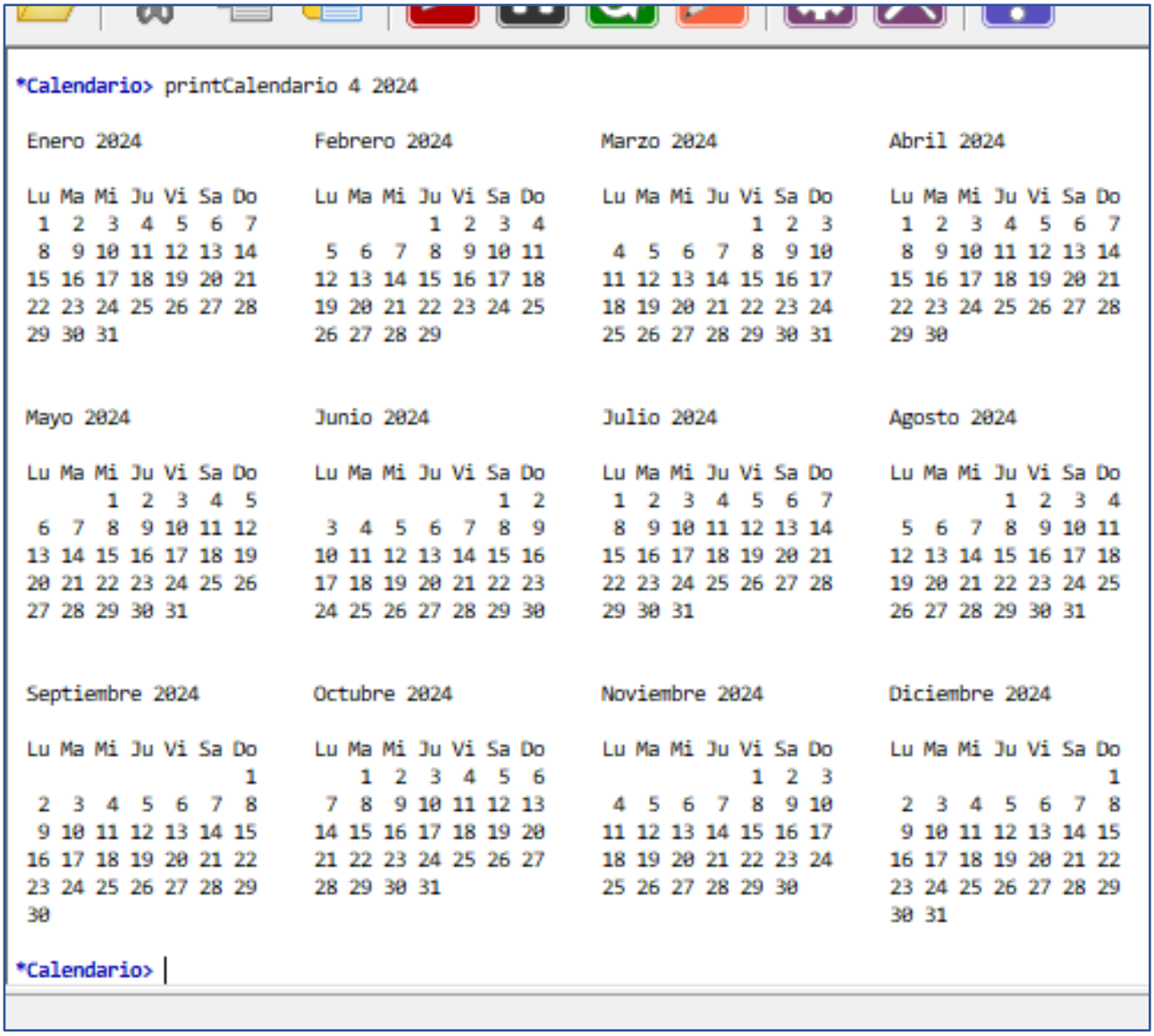
Programación Funcional 2024/2025

- Enunciado de Práctica 1 -

La práctica consiste escribir un programa en Haskell para crear el calendario de cualquier año e imprimirlo en pantalla en un número de columnas (3 ó 4).

Ejemplo Para obtener el calendario en 4 columnas del año 2024 evaluaremos la expresión:



```
*Calendario> printCalendario 4 2024

Enero 2024                Febrero 2024              Marzo 2024                Abril 2024

Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do
 1  2  3  4  5  6  7                1  2  3  4                1  2  3          1  2  3  4  5  6  7
 8  9 10 11 12 13 14       5  6  7  8  9 10 11       4  5  6  7  8  9 10       8  9 10 11 12 13 14
15 16 17 18 19 20 21      12 13 14 15 16 17 18      11 12 13 14 15 16 17      15 16 17 18 19 20 21
22 23 24 25 26 27 28      19 20 21 22 23 24 25      18 19 20 21 22 23 24      22 23 24 25 26 27 28
29 30 31                  26 27 28 29               25 26 27 28 29 30 31      29 30

Mayo 2024                 Junio 2024                Julio 2024                Agosto 2024

Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do
       1  2  3  4  5                   1  2       1  2  3  4  5  6  7                1  2  3  4
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       8  9 10 11 12 13 14       5  6  7  8  9 10 11
13 14 15 16 17 18 19      10 11 12 13 14 15 16      15 16 17 18 19 20 21      12 13 14 15 16 17 18
20 21 22 23 24 25 26      17 18 19 20 21 22 23      22 23 24 25 26 27 28      19 20 21 22 23 24 25
27 28 29 30 31            24 25 26 27 28 29 30      29 30 31                  26 27 28 29 30 31

Septiembre 2024           Octubre 2024              Noviembre 2024            Diciembre 2024

Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do      Lu Ma Mi Ju Vi Sa Do
                   1          1  2  3  4  5  6                1  2  3                   1
 2  3  4  5  6  7  8       7  8  9 10 11 12 13       4  5  6  7  8  9 10       2  3  4  5  6  7  8
 9 10 11 12 13 14 15      14 15 16 17 18 19 20      11 12 13 14 15 16 17       9 10 11 12 13 14 15
16 17 18 19 20 21 22      21 22 23 24 25 26 27      18 19 20 21 22 23 24      16 17 18 19 20 21 22
23 24 25 26 27 28 29      28 29 30 31               25 26 27 28 29 30         23 24 25 26 27 28 29
30                                                                            30 31

*Calendario>
```

Para desarrollar la práctica, se dispone de un módulo **Calendario** donde se proporcionan algunas funciones (ya definidas, o bien se da su especificación informal), de forma que dirigen una forma de abordar el problema a resolver.

El resto de funciones necesarias deben ser declaradas y definidas de forma clara y precisa, haciendo uso de los esquemas recursivos estudiados y demás características propias del paradigma funcional.

Obs: No se permite el uso del **let** ni del **$**

```
module Calendario where
type Dibujo = [Linea] -- cada dibujo es una lista de lineas
type Linea = [Char]   -- cada linea es una lista de caracteres
type Year = Int
type Columna = Int    -- es 3 o 4

-- Para imprimir un dibujo en pantalla:
printDibujo :: Dibujo -> IO()
printDibujo dib = do
          putStr "\n" -- putStr es la función que imprime un String
          (putStr . concat . map (++"\n")) dib

-- Imprime, con un numero de columnas, el calendario de un año:
printCalendario :: Columna -> Year -> IO()
printCalendario c a = printDibujo (calendario c a)
```

Función principal: **calendario**

```
-- Dibujo de un calendario (en c columnas) de un año dado:
calendario:: Columna -> Year -> Dibujo
calendario c = bloque c . map dibujomes . meses
```

donde:

1) La función **meses** devuelve, para un año dado, la información relevante de cada mes. Para el primer día del mes usaremos: 1=lunes, …, 7=domingo.

```
meses :: Year -> [(String, Year, Int, Int)]
-- meses n devuelve una lista de 12 elementos con los datos relevantes de cada uno de
--los meses del año n: (nombre_mes, n, primer_día_mes, longitud_mes)
```

Ejemplo:

```
*Calendario> meses 2024
[("Enero",2024,1,31),("Febrero",2024,4,29),("Marzo",2024,5,31),("Abril",2024,1,30),("May
o",2024,3,31),("Junio",2024,6,30),("Julio",2024,1,31),("Agosto",2024,4,31),("Septiembre"
,2024,7,30),("Octubre",2024,2,31),("Noviembre",2024,5,30),("Diciembre",2024,7,31)]
```

2) La función **dibujomes**, dada la información relevante del mes, devuelve su dibujo.

```
dibujomes ::(String, Year, Int, Int) -> Dibujo
-- dibujomes (nm,a,pd,lm) devuelve un dibujo de dimensiones 10x25 formado por el título
-- y la tabla del mes de nombre nm y año a.
-- Necesita como parámetros: pd=primer dia y lm=longitud del mes.
```

Ejemplo:

```
*Calendario> dibujomes ("Octubre",2024,2,31)
```

```
[" Octubre 2024            ","                          "," Lu Ma Mi Ju Vi Sa Do    ",
"     1  2  3  4  5  6    ","  7  8  9 10 11 12 13    "," 14 15 16 17 18 19 20    ",
" 21 22 23 24 25 26 27    "," 28 29 30 31             ","                         ",
"                         "]
```

```
*Calendario> printDibujo (dibujomes ("Octubre",2024,2,31))

 Octubre 2024

 Lu Ma Mi Ju Vi Sa Do
     1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29 30 31
```

de forma que si le aplicamos la función **printDibujo** quedará el dibujo del mes (dimensiones 10x25):

```
*Calendario> alto (dibujomes ("Octubre",2019,2,31))
10
*Calendario> ancho (dibujomes ("Octubre",2019,2,31))
25
```

3) La función **boque** nos permitirá agrupar los dibujos de una lista para hacer un solo dibujo

```
bloque :: Int -> [Dibujo] -> Dibujo
-- bloque n lisDib es el dibujo formado al agrupar de n en n los dibujos de lisDib,
--        extender cada sublista y luego apilar los resultados.
```

4) Entre el resto de funciones, se os proporciona **ene1** y se debe diseñar la función **pdias** que servirá para calcular los primeros días de la semana de cada mes de un año

```
ene1 :: Year -> Int
ene1 a = mod (a + div (a-1) 4 - div (a-1) 100 + div (a-1) 400) 7
-- ene1 a devuelve el dia de la semana del 1 de enero del año a
--    siendo 1=lunes, 2=martes, ..., 6=sabado, 0=domingo

pdias :: Int -> [Int]
-- pdias a devuelve una lista con 12 dias que son los dias de la
--     semana en que comienza cada mes del año a siendo
--     1=lunes, 2=martes, ..., 6=sabado y 7=domingo
```

Ejemplo:

```
*Calendario> pdias 2023
[7,3,3,6,1,4,6,2,5,7,3,5]
*Calendario> pdias 2024
[1,4,5,1,3,6,1,4,7,2,5,7]
*Calendario> pdias 2025
[3,6,6,2,4,7,2,5,1,3,6,1]
*Calendario>
```