Programación Declarativa
2007-2008
Práctica 3

1. Escriba una **expresión** en Caml cuyo valor sea siempre el mismo que:

   - **`abs x`** (* sin utilizar la función **`abs`** *)
   - **`not x`** (* sin usar el operador **`not`** *)
   - **`max x y`** (* sin usar la función **`max`** *)
   - **`min x y`** (* sin usar la función **`min`** *)
   - **`string_of_bool b`** (* sin utilizar la función **`string_of_bool`** *)
   - **`x mod y`** (* sin utilizar el operador **`mod`** *)
   - **`x && (y || z)`** (* sin usar los operadores **`&&`** y **`||`** *)
   - **`Char.lowercase c`** (* sin usar la función **`Char.lowercase`** *)

2. Escriba en ocaml **definiciones** "alternativas" para las funciones **`abs`**, **`not`**, **`max`**, **`min`**, **`string_of_bool y lowercase`** , como si no estuviesen predefinidas.

3. Intente deducir “a mano” cuál será el resultado de la compilación y ejecución de las siguientes frases (expresiones y definiciones) si se introdujesen en ese orden en el compilador . Escriba el resultado como lo escribiría el compilador interactivo de ocaml y comprube luego la respuesta real del compilador al introducir esas frases. Tome buena de todas las respuestas inesperadas y trate de explicar su razón. Si se produjera algún error, lea atentamente el mensaje del compilador y tome nota de por qué se ha producido.

```
let x, y = 1, 2;;

let x, y = y, x in x - y;;

x - y;;

let x = y and y = x in x - y;;

let x = y in let y = x in x - y;;

let z = x + y;;

let x = x + y and y = x - y;;

x - y;;

let x = 5;;

z;;

let y = 5 in x + y;;

x + y;;

let y = 3 * y in x + y;;
```

```
let z = let y = 3 * y in x + y;;

let x = x + y in let y = x + y in x + y;;

let y = let x = x + y in x + y in x + y;;

x + y + z;;
```

4. Las definiones de funciones de la forma

   ```
   let <name> = function <x> -> <exp>
   ```

   pueden abreviarse de la forma

   ```
   let <name> <x> = <exp>
   ```

   Así, por ejemplo, en vez de

   ```
   let doble = function x -> 2 * x
   ```

   podemos escriblir

   ```
   let doble x = 2 * x
   ```

   Utilice esta forma abreviada para reescribir las definiciones de las funciones **f**, **dup** y **absolut** que aparecían en el ejercicio 2 de la 2ª práctica.

5. En ocaml pueden hacerse definiciones recursivas si añadimos la palabra reservada "rec" justo después de "let":

   ```
   let rec fact n =
       if n <= 1 then 1 else n * fact (n-1);;

   let rec fib n =
       if n <= 1 then n
       else fib (n-1) + fib (n-2);;

   let rec sumalista l =
       if l = [] then 0
       else List.hd l + sumalista (List.tl l);;
   ```

   Defina una función "`nprimeros: int -> int list`" que devuelva, para cada número *n*, la lista de los primeros *n* números naturales. Así, por ejemplo, *nprimeros 4* deberá ser la lista *[1; 2; 3; 4]*.