TP 1 Programmation fonctionnelle

1 Environnement de travail

1. Installez ocaml sur votre machine en suivant les instructions : `https://webusers.i3s.unice.fr/~elozes/enseignement/PF/install-guide.html`
2. Comment lancer un toplevel ?
   (a) depuis un navigateur web, sans même avoir installé OCaml, grâce à TryOCaml `https://try.ocamlpro.com`.
   (b) depuis un terminal, il suffit de taper la commande ocaml.
   (c) en installant un toplevel non-officiel mais plus populaire : i) ocaml-top fait l'indentation automatique et la coloration syntaxique ii) utop offre la complétion automatique
   (d) Quelques remarques sur le toplevel
   — Avant d'afficher la valeur calculée, le toplevel affiche son type (int, string, float, ...)
   — On peut écrire sur plusieurs lignes, c'est ;; qui marque la fin de la saisie
   — La commande `#quit` interrompe le toplevel
   (e) A la place du toplevel, vous pouvez compiler votre fichier :

```
$ ocamlopt -o execfile file.ml
$ ./execfile
```

2 Exercices

1. Définissez la fonction `let add x y = x + y`. Laquelle des expresions suivantes produit un entier ? Testez vos réponses
   — `add 5 1`

   Solution. int □
   — `add 5`

   Solution. int -¿ int □
   — `(add 5) 1`

   Solution. int □
   — `add (5 1)`

   Solution. error □
2. Définissez une fonction qui prend un entier d et une chaîne de characters m en entrée et renvoie true uniquement lorsque d et m forment une date valide. Ici, une date valide a un mois qui est l'une des abréviations suivantes : Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sept, Oct, Nov, Dec. Et le jour doit être un nombre compris entre 1 et le nombre minimum de jours de ce mois, inclus. Par exemple, si le mois est Jan, alors le jour est

compris entre 1 et 31, inclus, tandis que si le mois est Feb, alors le jour est compris entre 1 et 28, inclus. À quel point pouvez-vous rendre votre fonction concise ?

Solution.

```
let date_fun (d:int) (m:string) =
   if d > 31 || d < 0 then false
   else if d <= 28
           && List.mem m ["Jan";"Feb";"Mar";"Apr";"May";"Jun";"Jul";
                          "Aug";"Sept";"Oct";"Nov";"Dec"] then true
   else if d <= 30
           && List.mem m ["Apr";"Jun";"Sept";"Nov"] then true
      else if d = 31
           && List.mem m  ["Jan";"Mar";"May";"Jul";"Aug";"Oct";"Dec"]
           then true
   else false
```

□

3. Définissez la fonction `val_abs` qui renvoie la valeur absolue d'un réel

Solution.

```
let val_abs x = if x < 0.0 then -.x else x;;
```

□

4. Définissez la fonction `signe` telle que `signe x` renvoie 0 si $x = 0$, 1 si $x > 0$, et -1 sinon

Solution.

```
let signe x =
  if x = 0 then 0 else
  if x > 0 then 1 else -1;;
```

□

5. Définissez la fonction `fac` qui calcule la factorielle de son argument

Solution.

```
let rec fac n =
  if n = 0 then 1 else n * fac (n - 1);;
```

□

6. Définissez la fonction `est_diviseur` telle que `est_diviseur n d` renvoie `true` si d divise n, false sinon

Solution.

```
let est_diviseur n d = n mod d = 0;;
```

□

7. Définissez la fonction `est_premier` qui indique si son argument est un nombre premier. Veillez à définir une fonction auxiliaire locale (une "sous-fonction").

Solution.

```
let est_premier n =
  let rec inter d =
  d * d > n || (not (est_diviseur n d) && inter (d + 1)) in
  inter 2;;
```

□

8. Définissez une fonction `plus a b` récursive , qui effectue la somme de a et b en utilisant l'idée que pour ajouter b, on additionne "b fois" la valeur 1.

Solution.

```
let rec plus a b =
  if a = 0 then b
  else 1 + plus (a-1) b
```

□

9. Définissez une fonction `prod a b` récursive, qui effectue le produit de a par b en utilisant l'idée que $a \times b$ revient à faire a + a + · · · + a + a (b fois).

Solution.

```
let rec prod a b =
  if a = 1 then b
  else plus b (prod (a-1) b);;
```

□