TD 4 Programmation fonctionnelle

1 Les types option

Problème : quoi faire lorsqu'une fonction ne peut pas renvoyer de réponse ?

Exercice 1 Implémenter une fonction qui trouve l'élément minimal dans une liste

Solution.

```
let rec list_min ls =
  match ls with
  | [a] -> a
  | hd :: tl -> min hd (list_min tl)
```

□

Comme nous avons déjà discuté, cette fonction ne marche pas (elle va lancer une exception `Match_failure`) si la liste est vide. De plus, le compilateur donnera un avertissement indiquant que la correspondance de motifs n'est pas exhaustive.

Exercice 2 Trouver l'index d'un élément v dans une liste ls. Les indices commencent à 0.

Solution.

```
let index v ls =
  let rec find i ls =
    match ls with
    | hd :: tl -> if hd = v then i else find (i+1) tl
    | [] -> (-1)  (* Peut-on mieux faire? *)
  in
  find 0 ls

let () =
  let v = 12 in
  let i = index v [10; 12; 45; 34] in
  if i = -1 then
    Printf.printf "Not found\n"
  else
    Printf.printf "Found at the index %i\n" i
```

□

Nous ne pouvons compter que sur la diligence du programmeur pour vérifier le code d'erreur -1 à chaque fois que la fonction est utilisée. Il est très probable que la fonction soit mal utilisée de temps en temps, ce qui peut entraîner le blocage du programme.

Une solution est d'utiliser le type d'option, qui est défini comme suit :

`type 'a option = None` Some of 'a—

Il peut prendre deux valeurs possibles :

— `None`, ce qui signifie : il n'y a pas de réponse,
— —Some v—, ce qui signifie : voici la réponse v , où v peut être n'importe quelle valeur OCaml.

Donc pour l'exercice précédent on pourrait écrire :

```
let index v ls =
  let rec find i ls =
    match ls with
    | hd :: tl -> if hd = v then Some i else find (i+1) tl
    | [] -> None
  in
  find 0 ls

let () =
  let v = 12 in
  match index v [10; 12; 45; 34] with
  | Some i -> Printf.printf "Found at the index %i\n" i
  | None -> Printf.printf "Not found\n"
```

Si la solution n'est pas trouvée, on retourne `None`, sinon, si la solution est x, on retourne `Some x`.

Étant donné que la fonction renvoie `int option` au lieu de simple int, ses valeurs de retour ne sont plus des nombres, elles prennent la forme de Some ou None et peuvent être distinguées à l'aide de la correspondance de motifs.

Exercice 3 Re-écrire l'exercice 1 un utilisant les types option :

Solution.

```
let rec list_min = function
  | [] -> None
  | hd::tl ->
      begin match list_min tl with
        | None -> Some hd
        | Some m -> Some (min hd m)
      end

let () =
  match list_min [10; 12; 45; 34] with
  | Some m -> Printf.printf "Smallest element is %i\n" m
  | None -> Printf.printf "The list is empty\n"
```

□

Étant donné que la valeur de retour est Some ou None, nous devons la faire correspondre dans l'appel récursif, nous devons donc utiliser la correspondance de motifs imbriqués.

Exercice 4 Re-écrire l'exercice 1 un utilisant les types option et utiliser une variable accumulateur, qui mémorise le plus petit élément jusqu'à présent :

Solution.

```
let list_min ls =
  match ls with
  | [] -> None
  | hd :: tl ->
      let rec find_min best ls =
        match ls with
        | [] -> best
        | hd :: tl -> find_min (min hd best) tl
      in
      Some (find_min hd tl)
```

□

Exercice 5. Trouvé la liste des racines carrées. Example :
`square_roots [4.0; -9.0; 5.0; -8.0] --> [Some 2.; None; Some 2.23606797749979; None]`
(Vous pouvez utiliser la fonction `sqrt : float -> float` pour calculer les racines carrées réelles.)

Solution.

```
let rec square_roots ls =
  match ls with
  | hd :: tl ->
      let x =
        if hd >= 0.0 then Some (sqrt hd) else None
      in
      x :: square_roots tl
  | [] -> []
```

□

Exercice 6. Trouver la valeur maximale parmi les éléments des deux listes. Exemples :

```
list_max2 [12; 35; 7] [18; 22] --> Some 35
list_max2 [] [22] --> Some 22
list_max2 [] [] --> None
```

Solution.

```
let rec list_max = function
  | [] -> None
  | hd::tl ->
      begin match list_max tl with
        | None -> Some hd
        | Some m -> Some (max hd m)
      end
```

```
let list_max2 ls1 ls2 =
  match list_max ls1, list_max ls2 with
  | Some x, Some y -> Some (max x y)
  | Some x, None -> Some x
  | None, Some y -> Some y
  | None, None -> None
```

□

Exercice 7. Trouver l'incrément positive maximale entre deux éléments consécutifs dans une liste. Renvoier `None` si aucune incrément n'est observée dans la liste, soit parce que la liste est trop courte, soit parce que toutes les modifications consécutives ne sont pas positives. Exemples :

```
max_incr [12; 35; 77; -72] --> Some 42  (* 35 -> 77 *)
max_incr [200; 100; 15] --> None        (* no increase *)
max_incr [123; 123; 123] --> None       (* no increase *)
max_incr [10] --> None                  (* no increase *)
```

Solution.

```
let rec max_incr ls =
  match ls with
  | a :: b :: tl ->
      let diff = b - a in
      let tl_answer = max_incr (b::tl) in
      if diff > 0 then
        match tl_answer with
        | Some mdiff -> Some (max diff mdiff)
        | None -> Some diff
      else
        tl_answer
  | _ -> None
```

□

Exercice 8. Définir un type de donné date comme étant une valeur de type int * int * int. Les exemples de triplet de type date incluent (5, 6, 2024) et (0, 0, 1000). Une date est un triplet de type date dont la première partie est un jour entre 1 et 31 (ou 30, 29 ou 28, selon le mois et l'année), la deuxième partie est un mois entre 1 et 12 et la troisième partie est une année positive (c'est-à-dire une année de l'ère commune), Donc . (5, 6, 2024) est une date ; (0, 0, 1000) ne l'est pas.

1. Ecrire trois fonctions pour extraire le jour, le mois, l'année.
2. Ecrire une fonction pour déterminer si une date est valide (Attention aux année bissextiles : les années sont bissextiles si elles sont multiples de quatre, mais pas si elles sont multiples de cent, à l'exception des années multiples de quatre cents qui, elles, sont également bissextiles.)

3. Écrire une fonction `is_before` qui prend deux dates en entrée et évalue à true ou false. Elle évalue à true si le premier argument est une date qui précède le deuxième argument. Si les deux dates sont identiques, le résultat est false. Utiliser les type option pour gérer aussi les dates non valides.

Solution.

```
type date = int*int*int;;

let today = (24,10,2024);;

let day d =
  match d with
    (x,_,_) -> x;;

let month d =
  match d with
    (_,m,_) -> m;;

let year d =
  match d with
    (_,_,y) -> y;;

let is_valid d =
  match d with
    (x,m,y) when (x<=0 || x>31 || m<=0 || m >12 || y<=0) -> false
  | (x,2,y) -> if (y mod 4==0) && ((y mod 100 <>0) || (y mod 400 ==0)) then x <=29
      else x<=28
  | (x,m,y) when (m == 4 || m == 6 || m==9 || m==11) -> x<=30
  | (x,m,y) -> x<=31;;

let is_before d1 d2 =
  if (is_valid d1) && (is_valid d2) then
    match (d1,d2) with
      ((x1,m1,y1),(x2,m2,y2)) ->
      Some (y1 < y2 || (y1 = y2 && m1 < m2) || (y1 = y2 && m1 = m2 && d1 < d2))
  else None
```

□