Révisions OCaml

MP2I Lycée Pierre de Fermat

Recoder les fonctions natives

Le module List d'OCaml contient de nombreuses fonctions utilitaires sur les listes. Vous pouvez consulter sa documentation ici : v2.ocaml.org/api/List.html.

Un bon entraînement est de recoder par vous même les fonctions de ce module. Pour chacune des fonctions suivantes, cherchez sa documentation sur le lien donné plus haut, et codez votre propre version. Vous pouvez comparer votre version avec celle d'OCaml pour vérifier. Pour utiliser une fonction du module List, il faut utiliser la syntaxe `List.nom_fonction`. Par exemple, pour la fonction `length` qui calcule le nombre d'éléments d'une liste : `List.length [2;3;4;5]`.

Q1. `length`

Q2. `mem`

Q3. `map`

Q4. `rev`

Q5. `init`

Q6. `fold_left`

Q7. `fold_right`

Q8. `iter`

Q9. `filter`

Q10. `split`

Q11. `assoc`

Q12. `exists`

Q13. `forall2`

Quiz

Cette partie est à faire sans utiliser d'interpréteur ou de compilateur, vous devez donc réfléchir à chaque question sans écrire de code. Lorsque vous êtes convaincu d'avoir la réponse, vous pouvez vérifier sur machine, mais il est important de vous entraîner à raisonner sans machine.

Trouver des expressions d'un type Pour chaque type suivant, trouver une expression de ce type. Lorsqu'un opérateur est précisé, vous devez l'utiliser dans votre expression

Q1. `int -> int`

Q2. `unit -> int`

Q3. `int -> int -> int` en utilisant `=`

Q4. `bool -> (int * int)` en utilisant `^`

Q5. `int list -> bool`

Q6. `'a -> 'a * 'a`

Q7. `'a * 'b -> 'b * 'a`

Q8. `'a -> 'b -> 'a`

Q9. `'a -> 'b -> int`

Q10. `('a -> 'b)-> ('b -> 'c)-> 'a -> 'c`

Q11. `('a -> 'b)-> 'a list -> 'b list`

Q12. `('a -> 'b)-> 'a list -> 'c list`

Q13. `( ('a -> 'b)list * ('a list))-> 'b list`

Q14. `'a -> ('a * 'b)list -> 'b`

Trouver le type d'une fonction Pour chacune des fonctions suivantes, donner son type.

```
let q15 x y = x + y

let q16 (x, y) = x + y

let q17 a b = (a, b)

let rec q18 l = match l with
  | [] -> ""
  | x :: q -> x ^ q18 q

let q19 f a = match f a with
  | (0, _) -> 0
  | (_, 0) -> 1
  | _ -> 2

let q20 f g h =
  f 0 + g 0 + h

let q21 x y z t w =
  w ((x y) (z t))

let rec q22 m p =
  let a, b, c = m 0 in
  if p a = p b then c
  else q22 m (fun x -> p x - 1)

let rec q23 a b =
  if a b = b then b
  else q23 a (a b)

let rec q24 a b c =
  match a with
  | [] -> c
  | x::q -> if x < b then q24 (b::q) x c else q24 ((c, 0)::q) x []
```

Trouver l'erreur Pour chacune des expressions suivantes, déterminer le type d'erreur (erreur de syntaxe ou bien erreur de typage) et l'expliquer :

```
let q25 =
  1.0 + 2.0

let q26 =
   let a = 1;
   let b = 2;
   a + b

let q27 =
  let li = [1; 2; 3] in "bla" :: li

let q28 =
  let li = [1;2;3] in li :: 4

let q29 =
  let rec f l =
  match l with
  | [] -> []
  | x :: q -> f x :: f q

let q30 u v = (u v, v 2, u 2)
```

Évaluer une expression Déterminez la valeur de chacune des expressions suivantes :

```
let q31 =
  let a = 2 in
  let b = a + 3 in
  let a = a + b in
  a * b

let q32 =
  let f x y = x * y x in
  let add x y = x + y in
  f 3 (add 5)


let q33 =
  let rec f la lb =
    match la with
    | [] -> lb
    | x :: q -> f q (x :: lb)
  in f [2; 3] [4; 5]

let q34 =
  let rec f la = match la with
    | [] -> failwith "Erreur"
    | [x] -> x
    | x::y::q -> f ((if x < y then x else y) :: q)
  in f [8;4;1;3;7]

let q35 =
  let rec f l = match l with
  | [] -> 0
  | [x] -> x
  | x :: y :: q -> x + f q
  in f [2;4;1;5;3]
```

Évaluer une expression (version dure) Déterminez la valeur de chacune des expressions suivantes :

```
1
2  let q36 =
3    let u c = String.make 1 c in
4    let rec f a b r i c e =
5      if b > c then
6        i r
7      else
8        let o = f a (b+1)
9        in o  (u a.[b] ^ r) (fun t -> i (t ^ u a.[c])) (c-1) e
10   in f "salutation" 0 "he" (fun s ->s) 9 0
11
12 let q37 =
13   let p a b x = x a b in
14   let g = p 5 3 in
15   let e c x = c (fun u v -> x 2 (u-v)) in
16   let s c = c (fun u v -> u) + c (fun u v -> v) in
17   s g * s (e g)
18
19 let q38 =
20   List.fold_right
21     (fun p l -> p::List.filter (fun a -> (a/p)*p <> a) l)
22     (List.init 20 ((+) 2))
23     []
24
25 (* Pour tester la dernière expression, vous devez lancer utop avec une
26    option désactivant certaines contraintes de types, en faisant:
27    utop -rectypes *)
28 let q39 =
29   let z f =
30     let g = fun x -> f(fun v-> x x v)
31     in g g
32   in let f = z (fun p n -> if n < 1 then [] else p (n-1) @ n :: p (n-1))
33   in f 3 ;;
```