Programmation Fonctionnelle
Types Algébriques

Adrien Durier

26 septembre 2023

Exercice : Types Algébriques et Formes Géométriques

Énoncé : Définir un type algébrique `forme` pour représenter des formes géométriques : cercles, rectangles et triangles.

1. Écrire la définition du type `forme`.
2. Écrire une fonction `périmètre : forme -> float`.
3. Écrire une fonction `aire : forme -> float`.

```
type forme =
```

Exercice : Types Algébriques et Formes Géométriques

Énoncé : Définir un type algébrique `forme` pour représenter des formes géométriques : cercles, rectangles et triangles.

1. Écrire la définition du type `forme`.
2. Écrire une fonction `périmètre : forme -> float`.
3. Écrire une fonction `aire : forme -> float`.

```
type forme =
  | Cercle of float
  | Rectangle of float * float
  | Triangle of float * float * float
```

Correction : Types Algébriques et Formes Géométriques

```
let périmètre f =
  match f with
  | Cercle r -> 2. *. 3.14159 *. r
  | Rectangle (w, h) -> 2. *. (w +. h)
  | Triangle (a, b, c) -> a +. b +. c

let aire f =
  match f with
  | Cercle r -> 3.14159 *. r *. r
  | Rectangle (w, h) -> w *. h
  | Triangle (a, b, c) ->
      let s = (a +. b +. c) /. 2. in
      sqrt (s *. (s -. a) *. (s -. b) *. (s -. c))
```

Le type option

Question : Comment définir un type qui permet de ne pas retourner de résultat en cas d'échec d'une opération ?

```
type 'a option =


```

Le type option

Question : Comment définir un type qui permet de ne pas retourner de résultat en cas d'échec d'une opération ?

```
type 'a option =
 | None
 | Some of 'a
```

- `None` représente l'absence de valeur.
- `Some x` représente une valeur présente.

(c'est dans la librairie standard d'Ocaml...)

Exercice : Utilisation du Type Option et Exceptions

Énoncé : Écrire deux fonctions pour trouver l'index d'un élément dans une liste.

1. `find_element : 'a -> 'a list -> int option`
   Retourne `Some index` si l'élément est trouvé, sinon `None`.
2. `find_element_exn : 'a -> 'a list -> int`
   Utilise `find_element` et lance une exception `Not_found` si l'élément n'est pas trouvé.

Correction : Utilisation du Type Option et Exceptions

```
let rec find_element x l =
  let rec aux i = function
    | [] -> None
    | y :: _ when x = y -> Some i
    | _ :: ys -> aux (i + 1) ys
  in aux 0 l
```

```
let find_element_exn x l =
  match find_element x l with
  | Some i -> i
  | None -> raise Not_found
```