

Cours 7 : Exceptions

1 Un mécanisme de gestion d'erreur

Il est fréquent d'écrire des fonctions partielles (au sens mathématique) comme la fonction `head` suivante :

```
# let head l = match l with [] -> ??? | x::l -> x;;  
val head : 'a list -> 'a = <fun>
```

Quelle peut être la valeur renvoyée par cette fonction quand `l=[]` ?

Une solution possible est de renvoyer une valeur de type `'a option` défini de la manière suivante :

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

Mais l'utilisation est alors alourdie par la nécessité de traiter le cas d'erreur de manière explicite et éventuellement en cascade :

```
# let ajouter_deux l = match head l with Some x -> Some(x + 2) | None -> None  
val ajouter_deux : int list -> int option = <fun>
```

Le mécanisme d'exception permet de traiter ce problème de manière élégante, en levant (`raise`) une *exception* dans le cas d'erreur.

```
# let head l = match l with [] -> raise Not_found | x::l -> x;;  
val head : 'a list -> 'a = <fun>
```

Ce mécanisme est transparent au typage, l'utilisation de cette fonction peut alors être écrite en ignorant les cas d'erreurs :

```
# let ajouter_deux l = head l + 2  
val ajouter_deux : int list -> int = <fun>
```

2 Définition et utilisation

Une exception est définie à l'aide du mot clé `exception`. À la manière d'un constructeur de type somme, une exception doit commencer par une majuscule et peut attendre des arguments.

```
# exception Mon_exception1;;  
# exception Mon_exception2 of string * int;;
```

Les exceptions sont des valeurs comme les autres : elles ont le type spécial `exn` et peuvent être passées en argument.

```
# Mon_exception1;;
- : exn = Mon_exception1
# Mon_exception2("toto",4);;
- : exn = Mon_exception2("toto",4)
```

La fonction prédéfinie `raise : exn -> 'a` permet de *lever* une exception afin d'interrompre le cours du programme.

```
# raise Mon_exception1; print_int 4;;
Exception: Mon_exception1
```

Une exception peut être *rattrapée* à l'aide de la construction `try with` qui permet de définir des gestionnaires d'exceptions.

```
# exception E of int;;
# let f x = if x=10 then raise (E(10)) else 10/x
# try f 10 with Division_by_zero -> 0 | E x -> x
```



Les arguments d'une exception ne peuvent pas être polymorphes.



Attention à l'imbrication des constructions `match with` et `try with`. On peut délimiter les blocs avec des `begin end` ou des parenthèses.

```
match ... with
| ... ->
  begin
    try ... with
      | ... -> ...
      | ... -> ...
    end
  | ... -> ...
```