# Programmation Fonctionnelle
# Cours 6
# Exceptions

October 19, 2012



## Table de matières





## Exceptions

- Interruption de l'exécution normale du programme.
- Se produit quand une fonction est appliquée à des valeurs non attendues (par exemple : division par zéro).
- Pas confondre avec les erreurs de typage !
- Toujours préférable à l'envoi des valeurs légales mais bidon.
- Arrêt du programme si l'exception n'est pas rattrapée.



## Exemples (predef1.ml)

```
(* Predefined exceptions *)

1/0;;
List.hd [];;
List.tl [];;
String.get "hello" 17;;
List.assoc "z" [("a", 5); ("b", 11)];;
```

## Exemples (exceptions1.ml)

```
(* Exceptions sont moins lourdes que des valeurs-er
type safe_int = Error | Int of int;;
let safe_div x y = if y=0 then Error else Int (x/y)
5 + safe_div 8 2;; (* erreur *)
let save_add x y = match x with
  | Error -> Error
  | Int x' -> match y with
      | Error -> Error
      | Int y' -> Int (x'+y')
;;
```



## Exemples (predef2.ml)

```
(* quel est le type d'une exception ? *)
Division_by_zero;;
Failure;;
Failure "toto";;
```



## Le type exn

- Les exceptions sont des constructeurs d'un type exn.
- Ces constructeurs peuvent avoir un argument.
- exn est donc comme un type somme, avec une seule différence importante :
- Le type exn est extensible : On peut définir de nouvelles exceptions, contrairement au vrais types sommes qui ne sont pas extensibles.



## Déclarer une exception

- Nouveau mot clef exception.
- Les noms des exceptions doivent commencer sur une majuscule.
- Les exceptions peuvent avoir un argument.
- Les exceptions ne peuvent pas être polymorphes.

## Exemples (declare1.ml)

```
exception Echec;;
exception Erreur_fichier of string;;
exception E1 of int list;;
exception E2 of 'a list;;  (* erreur *)
```



## Lever une exception

- Mot clef `raise`
- Le fait qu'une exception soit levée n'est pas pris en considération pour inférer un type.



## Exemples (declare2.ml)

```
(* exceptions et liaison statique *)

exception E of int;;
let f x = if x < 0 then raise (E x) else x+42;;
exception E of float;;
f 17;;
```



## Rattraper des exceptions

- `try` *e* `with` *filtrage-par-motifs*
- *filtrage-par-motifs* est une liste de cas (motif + expression), comme vue pour les filtrages par motif.
- Les motifs doivent être du type `exn`.
- Les expressions doivent être du même type, qui doit être égal au type de *e*.

## Exécution du `try e with`

1. Si l'évaluation de $e$ donne valeur $v$ : la valeur est $v$.
2. Si l'évaluation de $e$ lève une exception $x$ :
   - 2.1 Filtrage de $x$ par les motifs. Si un motif s'applique, l'expression correspondante est évaluée (rattrapage de $x$).
   - 2.2 Si aucun des motifs s'applique à $x$ : l'exception n'est pas rattrapée.
   - 2.3 L'évaluation d'une expression peut elle-même lever une exception.



## Exemples (catch1.ml)

```
let dico = [("a", "un"); ("wonderful", "formidable")
            ("is", "est")];;

let translate_mot m =
  try List.assoc m dico
  with Not_found -> m;;

let rec translate_phrase  = List.map translate_mot;;

translate_phrase ["Caml";"is";"wonderful"];;
```



## Exemples (catch2.ml)

```
(* erreur de typage *)

let f x y =
  try x/y with
    | Division_by_zero -> "Erreur"
;;
```



## Exemples (catch3.ml)

```
exception E of int;;
let f x = if x=0 then raise (E 0) else
  if x<10 then raise (E 1) else
    if x < 20 then raise (E 2)
    else x;;
let g x =
  try (string_of_int (f x)) with
    | E 0 -> "zero"
    | E 1 -> "petit"
;;

g 0;;
g 5;;
g 17;; (* exception pas rattrape *)
g 42;;
```

## Exemples (catch4.ml)

```
let f x =
  try x with
    | Division_by_zero -> -1
;;

(* Quel est le resultat ? *)
f (17/0);;
```



## Exemple : Test de complétude d'un arbre binaire

Un arbre binaire est dit *complet* (ou, *équilibré*) si

- Il est une feuille ;
- ou bien si ses deux fils sont
  - complets,
  - et de même hauteur.



## Exemples (complet1.ml)

```
(* sans exceptions, avec parcours multiples *)
type arbre = F | N of arbre * arbre;;

let rec hauteur a =
  match a with
  | F -> 0
  | N(g,d) -> 1 + max (hauteur g) (hauteur d);;

let rec complet a =
  match a with
  | F -> true
  | N(g,d) -> (complet g) && (complet d)    (* parco
              && (hauteur g = hauteur d);;  (* parco


complet (N(N(F,F),N(F,F)));;
complet (N(N(F,F),N(F,N(F,F))));;
```



## Exemples (complet2.ml)

```
(* avec exceptions et un seul parcours de l'arbre :
exception Incomplet

let complet a =
  let rec haut_aux a = match a with
    | F -> 0
    | N(g,d) ->
      let hg = haut_aux g
      and hd = haut_aux d in
      if hg = hd then (1 + hg) else raise Incomplet
  in try
    let _ = haut_aux a in true
  with Incomplet -> false;;

complet (N(N(F,F),N(F,F)));;
complet (N(N(F,F),N(F,N(F,F))));;
```

## Assertions

- Mot clef `ssert`, suivi d'une expression booléenne
- Lève une exception `Assert_f ilure` quand l'expression donne false, avec nom du fichier, numéro de ligne et colonne.
- Utile pour déclarer des invariants.
- Utile pour tester des invariants pendant l'exécution.
- On peut désactiver les assertions pendant la compilation d'un programme.
- Premier langage avec assertions : `Eiffel`.



## Exemples (assert1.ml)

```
let f x =
  assert (x>=0.);
  sqrt x
;;

f 2.;;
f (-5.);;
```



## Exemples (assert2.ml)

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

fac 4;;
fac (-4);;
```



## Types de requettes OCaml comprises par l'interpréteur

- Expression
- Liaison `let x = <expression>`
- Définition d'un type
- Définition d'une exception
- Directive pour l'interpréteur (comme `#tr ce f`)
- Des autres types de requettes viennent du système de modules et du système d'objets.