Les exceptions

Une exception est un mécanisme apparaissant lorsque le calcul d'une fonction aboutit à un échec. En cas d'erreur, OCaml déclenche une exception.
Par exemple, avec la fonction suivante : `let f t = t.(3) ;;`
Si on calcule `f [|0;1|]`, Caml déclenche une exception et renvoie le message d'erreur suivant :
`Exception: Invalid_argument "index out of bounds".`

OCaml possède un type exception, noté `exn`. Il renvoit donc un résultat de type exception lorsqu'un calcul échoue.
L'utilisateur peut créer lui-même de nouvelles exceptions, constantes ou paramétrées, avec la syntaxe suivante :
Exception constante : `exception Truc`
Exception paramétrée (ici par un entier) : `exception Truc of int`.

On notera que le nom d'une exception commence par une majuscule.
Certaines exceptions existent déjà en OCaml, par exemple `Not_found`, `Failure of string`, `Out_of_memory`.
On peut voir une liste des exceptions prédéfinies dans le manuel de l'utilisateur à l'url suivante :
https://caml.inria.fr/pub/docs/manual-ocaml/core.html

L'utilisateur peut déclencher une exception, et même rattraper une exception.
Pour déclencher une exception dans un programme, on écrit : `raise <nom de l'exception>`.

Exemple : La fonction factorielle, codée récursivement, ne renvoie un résultat cohérent que pour des entiers compris entre 0 et 33, car au-delà, on dépasse `max_int`, entier naturel maximal géré en OCaml, et on repasse dans les négatifs donc le résultat obtenu est incohérent. On souhaite donc déclencher une exception dans ce cas.

```
exception Overflow ;;
let rec facto n = match n with
   0 -> 1
 | _ when n>33 -> raise Overflow
 | _ ->  n * facto (n-1) ;;
```

Réponse de OCaml :

```
# facto 10 ;;
- : int = 3628800
# facto 34 ;;
Exception: Overflow.
```

En abrégé, pour déclencher `Failure` avec la chaîne "nom", on peut écrire : `failwith "nom"`.

Exemple : calcul de la factorielle avec exception renseignée pour les nombres négatifs :

```
let rec facto x =
if x = 0 then 1 else
if x < 0 then failwith "nb négatif"
else x * facto(x-1) ;;
```

Il est possible de rattraper une ou plusieurs exceptions, c'est-à-dire d'exécuter un bloc d'instructions lorsqu'une exception est déclenchée. La syntaxe est la suivante :

```
try            <instructions>
with  <exception 1> → <instructions>
                    ...
      <exception n> → <instructions>
```

Un des intérêts des exceptions est de pouvoir interrompre une boucle `for` lorsqu'on obtient un certain résultat et donc d'éviter le recours à une boucle `while` (parfois plus lourde à écrire).

Exemples :

- La fonction `div` déclenche une exception quand on divise par 0.

  Ayant défini un type entier comportant l'infini, la fonction `div2` renvoie le reste de la division de `x` par `y` si `y` est non nul et infini sinon.

```
exception Division_par_0 ;;
type entier = I of int | Infini ;;
let div x y = if y <>0 then x/y else raise Division_par_0 ;;
let div2 x y =
    try
    if y <>0 then I(x/y) else raise Division_par_0
    with Division_par_0 -> Infini ;;
```

- La fonction `test` renvoie la position du premier 0 d'un vecteur et -1 s'il n'y en a pas. Elle utilise une exception lui permettant de sortir de la boucle `for` lorsqu'on rencontre un 0.

  La fonction `testbis` en est une version sans exception avec une boucle `while`.

```
exception Problem of int  ;;
(* test v renvoie la position du premier 0 du vecteur v et -1 s'il n'y en a pas *)
let test v =
  try
  let n = Array.length v in
  for i = 0 to n-1 do
     if v.(i) = 0 then raise (Problem i);
  done;
  -1
  with  Problem i -> i
;;
let testbis t =
 let i = ref 0 and ok = ref false in
 while !i < Array.length t && not(!ok) do
  if t.(!i) = 0 then ok := true else incr i
 done;
 if !ok then !i else -1 ;;
```