Arguments optionnels et variantes.

Didier Rémy
2001 - 2002

http://cristal.inria.fr/~remy/mot/21/
http://www.enseignement.polytechnique.fr/profs/informatique/Didier.Remy/mot/21/

Cours	**Exercices**
1. Arguments nommés 2. Arguments optionnels 3. Variantes	1. Arguments optionnels

Slide 1

Voir aussi le tutorial.

NB : À partir de la version 3.03 il faut ouvrir le module `StdLabels` avec la commande

```
open StdLabels;;
```

avant pour exécuter les examples de ce cours.

Arguments nommés

Slide 2

Ocaml permet de nommer les arguments des fonctions. Cela peut déjà être observé sur le type des fonctions de librairie :

```
List.fold_left;;
```

```
- : f:('a -> 'b -> 'a) -> init:'a -> 'b list -> 'a = <fun>
```

Le type ci-dessus indique que les deux premiers arguments de la fonctions peuvent être nommés avec les étiquettes `f` et `init`, ce qui s'écrit ainsi (les étiquettes d'argument sont préfixée par `~`) :

```
List.fold_left ~f:(fun x l -> x * l) ~init:1 [2;3;4];;
```

Par défaut, les arguments nommés sont facultifs, *i.e.* ils peuvent ne pas être nommés, mais doivent être passés dans l'ordre. On peut aussi écrire :

```
List.fold_left (fun x l -> x * l) 1 [2;3;4];;
```

mais pas (sauf pour la version est 3.03 ou plus récente)

```
List.fold_left ~init:1 ~f:(fun x l -> x * l) [2;3;4];;
```

Définir des fonctions avec arguments nommés

Slide 3

On peut aussi, bien sûr, définir de nouvelles fonctions avec des arguments nommés :

```
let norm ~x:u ~y:v = u*u + v*v;;
```

```
val norm : x:int -> y:int -> int = <fun>
```

Il existe un racourci ou le nom de la variables est celui de l'étiquette de l'argument :

```
let norm ~x ~y = x*x + y*y;;
```

est équivalent à :

```
let norm ~x:x ~y:y = x*x + y*y;;
```

Étiquettes obligatoires et commutativité

On peut inverser ce comportement, *i.e.* rendre les arguments optionnels obligatoires, mais leur l'ordre non stricte en utilisant l'option `-labels` (aussi bien pour le toplevel que pour le compilateur).

Un des intérêts des arguments nommés est justement de ne plus avoir à se souvenir de l'ordre dans lequel ils doivent être passés.

Slide 4

Note *À partir de la version 3.03, l'option* `-labels` *n'existe plus. Le choix du mode étiquetté se fait en ouvrant ou pas le module* `stdLabels`*.*

Arguments optionnels

Un autre avantage des arguments nommés, y compris dans la version classique, *i.e.* sans l'option `-labels`est de rendre certains arguments optionnels, en fournissant lors de la définition de la fonction une valeur par défaut.

```
let incr ?pas:(x = 1) r = r := !r + x;;
```

```
val incr : ? pas:int -> int ref -> unit = <fun>
```

Slide 5

La fonction incrément reçoit un premier argument nommé avec l'étiquette `pas` qui est optionnel avec la valeur par défaut `1` pour l'argument de nom pas. On peut alors écrire :

comme auparavant :

```
let r = ref 0;;
incr r; !r;;
```

```
- : int = 1
```

ou bien

```
incr ~pas:10 r; !r;;
```

```
- : int = 11
```

Test sur la présence de l'argument

Un argument optionnel est un argument du type `'a option`... avec un peu de sucre syntaxique.

Par exemple, la définition

```
let incr ?pas:(x = 1) r = r := !r + x;;
```

est équivalente à

Slide 6

```
let incr ?pas:x_option r =
  let x = match x_option with Some x -> x | None -> 1
  in r := !r + x;;
```

Exercice 1 (Argument option(nel)) *Écrire l'exemple ci-dessus sans nommer l'argument en utilisant seulement le type option. Écrire les deux formes d'appel.* Réponse □

Type variantes

Les types concrets doivent être déclarés.

Un constructeur n'appartient qu'à un seul type concret.

Un type variante est un grand type somme dont les constructeurs sont prédéfini (un identificateur préfixé par un accent grave ‘) mais dont le domaine est typé plus finement.

Slide 7

```
let un = ‘Int 1 and demi = ‘Float 0.5;;
```

```
val un : [> ‘Int of int] = ‘Int 1
val demi : [> ‘Float of float] = ‘Float 0.500000
```

Par exemple, le type de `Demi` signifie que `demi` est construit avec l'étiquette `‘Float` de type `float` et peut être mélangé à des valeurs construites avec d'autres étiquettes.

Structures hétérogènes

Une liste hétérogène est donc possible :

```
let x = [ ‘Int 1; ‘Float 0.5 ];;
```

```
val x : [> `Int of int | `Float of float ] list
    = [`Int 1; `Float 0.500000]
```

La valeur `x` est une liste de valeur construite avec l'étiquette `` `Float `` portant des valeurs de type `float`, avec l'étiquette `` `Int `` portant des valeurs de type `int`, ou avec d'autres étiquettes.

Slide 8

Analyse par cas

```
let float_of_num z =
  match z with
  | `Int x -> float x
  | `Float x -> x;;
```

```
val float_of_num : [< `Int of int | `Float of float ] -> float = <fun>
```

Slide 9

L'argument de `float_of_num` est une valeur construite au plus avec l'étiquette `` `Float `` portant des valeurs de type `float` ou avec l'étiquette `` `Int `` portant des valeurs de type `int`.

1 Solutions des exercices

Exercice 1, page 6

```
let incr x_option r =
  let x = match x_option with Some x -> x | None -> 1
  in r := !r + x;;
let r = ref 0;;
incr None r; !r;;
```

```
val r : int ref = {contents=0}
```

```
incr (Some 10) r; !r;;
```

```
- : int = 10
```