Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.

Champ mutable d'un enregistrement

Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

Champ mutable d'un enregistrement

Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

Champ mutable d'un enregistrement

On peut déclarer en OCaml un enregistrement ayant des champs mutables grâce au mot-clé `mutable`.

Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

Champ mutable d'un enregistrement

On peut déclarer en OCaml un enregistrement ayant des champs mutables grâce au mot-clé `mutable`. Par exemple,

```
1  type variable = {mutable valeur : int};;
```

Introduction

- Pour le moment, nous nous sommes limités aux aspects fonctionnels d'OCaml et donc aux variables et aux structures de données non mutables.
- Cependant, OCaml est un langage de programmation multi-paradigme et la programmation impérative (et donc les variables mutables) peuvent être manipulées en OCaml.

Champ mutable d'un enregistrement

On peut déclarer en OCaml un enregistrement ayant des champs mutables grâce au mot-clé `mutable`. Par exemple,

```
1  type variable = {mutable valeur : int};;
```

⚠ Pour modifier la valeur du champ on utilise `<-`. Le symbole = est réservé à la comparaison.

Exemple

```
1  (* Type "variable"  ayant un champ valeur mutable*)
2  type variable = {mutable valeur : int};;
3
4  (* on crée une variable ayant son champ valeur à 42 *)
5  let u = {valeur = 42};;
6
7  (* comme ce champ est mutable, on change la valeur avec <-*)
8  u.valeur <- 20;
9  (* cette expression renvoie unit et modifie la valeur de u*)
10
11 (* Affichera 20 *)
12 print_int u.valeur
```

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement `let a = ref 5;;`.

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement `let a = ref 5;;`.
  On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement `let a = ref 5;;`.
  On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec `!`. On écrira par exemple `print_int !a` pour afficher le contenu du champ mutable de a.

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement `let a = ref 5;;`.
  On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec `!`. On écrira par exemple `print_int !a` pour afficher le contenu du champ mutable de a.
  On retiendra que c'est identique à `print_int a.contents`

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement `let a = ref 5;;`.
  On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec `!`. On écrira par exemple `print_int !a` pour afficher le contenu du champ mutable de a.
  On retiendra que c'est identique à `print_int a.contents`
- Pour modifier la valeur de a, une syntaxe plus simple est aussi disponible avec `:=`. On écrira par exemple `a := 5`

Les références

- Le type `ref` est prédéfini dans OCaml et correspond exactement à ce que nous venons de faire (sauf que le champ mutable s'appelle `contents`)
  Par exemple `let a = {contents = 5};` crée une variable ayant son champ mutable `contents` qui vaut 5.
- Afin d'alléger la syntaxe, on peut écrire directement `let a = ref 5;;`.
  On retiendra que l'effet reste le même : on a crée une variable a ayant un champ mutable entier qui contient 5.
- Si on veut accéder à la valeur du champ mutable, une syntaxe "adoucie" est aussi disponible avec `!`. On écrira par exemple `print_int !a` pour afficher le contenu du champ mutable de a.
  On retiendra que c'est identique à `print_int a.contents`
- Pour modifier la valeur de a, une syntaxe plus simple est aussi disponible avec `:=`. On écrira par exemple `a := 5`
  Par exemple, `a:= !a +1` permet d'incrémenter de 1 la valeur (du champ mutable) de a.

Exemple

Créer une reférence a vers 42 et une référence b vers 2023. Echanger le contenu de ces deux variables en utilisant une troisième référence `temp`

Exemple

Créer une reférence a vers 42 et une référence b vers 2023. Echanger le contenu de ces deux variables en utilisant une troisième référence temp

```
1 let a = ref 42;;
2 let b = ref 2023;;
3 let temp = ref !a;;
4 a := !b;
5 b := !temp;
```