

Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 5 - Sequence 3: Mutable data structures: mutable fields in records



Revisiting the records

We have met the record data structure in the course on Week 2 - Sequence 2.

- ▶ Records are tuples with *distinct named* components

- ▶ A typical *record type* declaration:

```
type some_type_identifier =  
  { field_name_1 : some_type_1; ...; field_name_n : some_type_n }
```

- ▶ A typical *record* definition:

```
let r = { field_name_1 = e1; ...; field_name_n = en }
```

Two dimensional points I

(* 2D points *)

```
type point2D = { x : int; y : int };;
```

```
# type point2D = { x : int; y : int; }
```

```
let origin = { x = 0; y = 0 };;
```

```
# val origin : point2D = {x = 0; y = 0}
```

(* create a new point at offset of given one *)

```
let offset_h p dx = {p with x=p.x+dx};;
```

```
# val offset_h : point2D -> int -> point2D = <fun>
```

```
let offset_v p dy = {p with y=p.y+dy};;
```

```
# val offset_v : point2D -> int -> point2D = <fun>
```

(* no modification is made to the original point *)

```
let p = offset_h origin 10;;
```

```
# val p : point2D = {x = 10; y = 0}
```

Two dimensional points II

```
origin;;
```

```
# - : point2D = {x = 0; y = 0}
```

Revisiting the records: mutable fields

We can declare *selected* field records as **mutable**.

```
type some_type_identifier =  
  { field_name_1 : some_type_1;  
    ...;  
    mutable field_name_i : some_type_i;  
    ...;  
    field_name_n : some_type_n }
```

The fields declared *mutable* can be modified in place.
For this, we use again the `<-` operator.

Colored movable two dimensional points I

(* RGB colors *)

```
type color = {r: int; g:int; b:int};;
```

```
# type color = { r : int; g : int; b : int; }
```

```
let black = {r=255;g=255;b=255};;
```

```
# val black : color = {r = 255; g = 255; b = 255}
```

(* movable colored 2D points *)

```
type point2D = { mutable x : int; mutable y : int ; c: color};;
```

```
# type point2D = {
```

```
  mutable x : int;
```

```
  mutable y : int;
```

```
  c : color;
```

```
}
```

Colored movable two dimensional points II

```
let origin = { x = 0; y = 0 ; c=black};;  
# val origin : point2D =  
  {x = 0; y = 0; c = {r = 255; g = 255; b = 255}}
```

(* create a new point at offset of given one *)

(* thanks to "with" we keep the same code *)

```
let offset_h p dx = {p with x=p.x+dx};;  
# val offset_h : point2D -> int -> point2D = <fun>  
let offset_v p dy = {p with y=p.y+dy};;  
# val offset_v : point2D -> int -> point2D = <fun>
```

(* no modification is made to the original point *)

```
let p = offset_h origin 10;;  
# val p : point2D =  
  {x = 10; y = 0; c = {r = 255; g = 255; b = 255}}
```

Colored movable two dimensional points III

```
origin;;  
# - : point2D =  
{x = 0; y = 0; c = {r = 255; g = 255; b = 255}}
```

(* start moving things around *)

```
let move p dx dy = p.x <- p.x+dx; p.y <- p.y+dy;;  
# val move : point2D -> int -> int -> unit = <fun>
```

(* p is modified *)

```
p;;  
# - : point2D =  
{x = 10; y = 0; c = {r = 255; g = 255; b = 255}}
```


Colored movable two dimensional points IV

```
move p 2 2;;
```

```
# - : unit = ()
```

```
p;;
```

```
# - : point2D =
```

```
{x = 12; y = 2; c = {r = 255; g = 255; b = 255}}
```

```
move p (-1) (-1) ;;
```

```
# - : unit = ()
```

```
p;;
```

```
# - : point2D =
```

```
{x = 11; y = 1; c = {r = 255; g = 255; b = 255}}
```

Summary

- ▶ Selected fields of a record type can be declared *mutable*
- ▶ The update operator `<-` modifies in place these mutable fields