- Introduction and overview
- Basic types, definitions and functions
- Basic data structures
- More advanced data structures
- Higher order functions
- **Exceptions, input/output and imperative constructs**

Table of Contents

Imperative features in OCaml

Getting and handling your Exceptions
Week 5 Echéance le déc 12, 2016 at 23:30 UTC

Getting information in and out
Week 5 Echéance le déc 12, 2016 at 23:30 UTC

Sequences and iterations
Week 5 Echéance le déc 12, 2016 at 23:30 UTC

Mutable arrays
Week 5 Echéance le déc 12, 2016 at 23:30 UTC

Mutable record fields
Week 5 Echéance le déc 12, 2016 at 23:30 UTC

Variables, aka References
Week 5 Echéance le déc 12, 2016 at 23:30 UTC

- Modules and data abstraction
- Project

PRODUCING FINE ASCII ART (175/175 points)

In this exercise, we will display black and white images as text, where a black dot is printed as a `'#'` and a white dot as a `' '`.

Instead of using imperative constructs for storing our images, images will simply be functions that take an `x` and a `y` and return a boolean that indicates if the function is black or white at this point.
This is materialized by the `image` type alias given in the prelude.
We will only use imperative features to display them.

1. Define a higher order function `display_image: int -> int -> image -> unit` that takes an integer `width`, another one `height`, a function which takes an `x` and a `y`, and renders (prints) the boolean function as a series of lines, using two nested `for` loops.
   Each line corresponds to a `y`, the first line printed being for `y = 0`, the last one for `y = height`.
   In each line, the first character printed must be for `x = 0`, the last one for `x = width`.
   When the function result is `true`, a sharp (`'#'`) must be displayed, and a space otherwise.
   To try your function, the prelude defines sample images and image builders.
   For instance, the image `disk 5 5 5` would be displayed as the following ASCII art, when rendered between coordinates `0 <= x <= 10` and `0 <= y <= 10`.

```
     #
   #######
  #########
  #########
  #########
 ###########
  #########
  #########
  #########
   #######
     #
```

2. Now, we want to blend images to compose complex images from simple ones. For this, we will use the `blend` type given the prelude.
   If we take two functions `f` and `g`, we have that:

   - `Image f`
     is the blended image looking exactly like `f`.
   - `And (Image f, Image g)`
     is the blended image that is black only where both `f` and `g` are both black.
   - `Or (Image f, Image g)`
     is the blended image that is black wherever either `f` or `g` or both are black.
   - `Rem (Image f, Image g)`
     is the blended image that is black wherever `f` is black and `g` is not.

   Write a recursive `render : blend -> int -> int -> bool` function, that tells, for a given `x` and `y` the boolean color of the point, considering the given blended image.

3. Define a function `display_blend: int -> int -> blend -> unit` that takes a `width`, another one `height`, a blended image, and displays it by combining the two previous functions.
   As an example, the blend `display_blend 10 10 (Rem (Image all_black, Image (disk 5 5 5)))` would be displayed as the following ASCII art.

```
#         #
#         #

#         #
#         #
#         #
##       ##
##### #####
```

Bonus question: Did you see that the type of `render` is actually equivalent to `blend -> image`?

THE GIVEN PRELUDE

```
type image = int -> int -> bool ;;

let all_white = fun x y -> false ;;

let all_black = fun x y -> true ;;

let checkers = fun x y -> y/2 mod 2 = x/2 mod 2 ;;

let square cx cy s = fun x y ->
  let minx = cx - s / 2 in
  let maxx = cx + s / 2 in
  let miny = cy - s / 2 in
  let maxy = cy + s / 2 in
  x >= minx && x <= maxx && y >= miny && y <= maxy
;;

let disk cx cy r = fun x y ->
  let x' = x - cx in
  let y' = y - cy in
  (x' * x' + y' * y') <= r * r
;;

type blend =
  | Image of image
  | And of blend * blend
  | Or of blend * blend
  | Rem of blend * blend
;;
```

YOUR OCAML ENVIRONMENT

```
let display_image width height f_image =
  for y = 0 to height do
    for x = 0 to width do
      if f_image x y = true then print_string "#" else print_string " "
    done
            ; print_string "\n"
  done
;;

let rec render blend x y = match blend, x, y with
  | Image im, x, y -> im x y
  | And (b1,b2), x, y -> (render b1 x y) && (render b2 x y)
  | Or (b1,b2), x, y -> (render b1 x y) || (render b2 x y)
  | Rem (b1,b2), x, y -> (render b1 x y) && not (render b2 x y)
;;

let display_blend width height blend =
  display_image width height (render blend)
;;
```

Evaluate >

Switch >>

Typecheck

Reset Templ

Full-screen

Check & Sa

v Exercise 1: `display_image` Completed, 55 pts

Found a toplevel definition for `display_image` .

You used a two nested loops, bravo!! 5 pts

Now I will check that it behaves correctly

Found `display_image` with compatible type.

Computing `display_image 10 10 all_black`

Expected output 5 pts

```
  ###########
  ###########
  ###########
  ###########
  ###########
  ###########
  ###########
  ###########
  ###########
  ###########
  ###########
```

Computing `display_image 10 10 all_white`

Expected output 5 pts

Computing `display_image 10 10 (square 7 3 3)`

Expected output 5 pts

```
        ###
        ###
        ###
```

Computing `display_image 10 10 (disk 3 7 3)`

Expected output 5 pts

```
     #
   #####
   #####
  #######
   #####
   #####
     #
```

Computing `display_image 10 10 checkers`

Expected output 5 pts

```
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
```

Computing `display_image 10 10 (disk 7 3 3)`

Expected output 5 pts

```
         #
       #####
       #####
      #######
       #####
       #####
         #
```

Computing `display_image 10 10 (square 7 3 3)`

Expected output 5 pts

```
        ###
        ###
        ###
```

Computing `display_image 10 10 (square 5 5 5)`

Expected output 5 pts

```
     #####
     #####
     #####
     #####
     #####
```

Computing `display_image 10 10 all_white`

Expected output 5 pts

Computing `display_image 10 10 (disk 5 5 3)`

Expected output 5 pts

```
        #
      #####
      #####
     #######
      #####
      #####
        #
```

v Exercise 2: `render` Completed, 20 pts

Found `render` with compatible type.

Computing

```
  render
    (Or
      (Rem (And (Image (square 5 5 5), Image (disk 5 5 5)),
        Or (Image (square 5 5 5), Image all_black)),
      Or (Rem (Image all_black, Image (disk 5 5 5)), Image checkers)))
    7
    6
```

Correct value `true` 1 pt

Computing

```
  render
    (Or
      (Rem (And (Image (square 5 5 5), Image all_black),
        And (Image (square 5 5 5), Image (disk 5 5 5))),
      Image checkers))
    6
    7
```

Correct value `true` 1 pt

Computing

```
  render
    (Or (Rem (Rem (Image all_black, Image (disk 5 5 5)), Image checkers),
      And (Or (Image checkers, Image all_black), Image (disk 5 5 5))))
    6
    6
```

Correct value `true` 1 pt

Computing

```
  render
    (Rem (Or (Rem (Image (disk 5 5 5), Image all_black), Image checkers),
      And (And (Image (square 5 5 5), Image (square 5 5 5)), Image checkers)))
    6
    7
```

Correct value `false` 1 pt

Computing

```
render (Rem (Image checkers, Rem (Image (disk 5 5 5), Image all_black))) 6 7
```

Correct value `true` 1 pt

Computing

```
  render
    (And (Image (disk 5 5 5),
      Or (Or (Image all_black, Image checkers), Image (square 5 5 5))))
    7
    6
```

Correct value `true` 1 pt

Computing

```
  render
    (Rem (Rem (Image checkers, Rem (Image checkers, Image (disk 5 5 5))),
      Image (square 5 5 5)))
```

```
Computing
  render
    (And (Image (square 5 5 5),
      And (And (Image (disk 5 5 5), Image all_black), Image (square 5 5 5))))
    6
    7
```

Correct value `true` 1 pt

Computing `render (Image (square 5 5 5)) 6 6`

Correct value `true` 1 pt

```
Computing
  render
    (Or (And (Image (disk 5 5 5), Image (disk 5 5 5)),
      And (Image checkers, Rem (Image (square 5 5 5), Image (disk 5 5 5)))))
    6
    7
```

Correct value `true` 1 pt

Computing `render (Image (disk 5 5 5)) 7 7`

Correct value `true` 1 pt

Computing `render (Image checkers) 6 7`

Correct value `true` 1 pt

```
Computing
  render
    (Rem (Or (Image all_black, Or (Image all_black, Image (square 5 5 5))),
      Rem (And (Image (square 5 5 5), Image (disk 5 5 5)),
       Rem (Image all_black, Image checkers))))
    7
    6
```

Correct value `false` 1 pt

```
Computing
  render
    (Or (And (Image all_black, Or (Image (square 5 5 5), Image (disk 5 5 5))),
      Image (disk 5 5 5)))
    7
    7
```

Correct value `true` 1 pt

```
Computing
  render
    (Rem (Image all_black,
      Or (Image (disk 5 5 5), And (Image checkers, Image (disk 5 5 5)))))
    7
    6
```

Correct value `false` 1 pt

```
Computing
  render
    (Rem (Or (Image (square 5 5 5), Image checkers), Image (square 5 5 5)))
    6
    6
```

Correct value `false` 1 pt

Computing `render (Image (disk 5 5 5)) 6 7`

Correct value `true` 1 pt

```
Computing
  render
    (Or (Image checkers,
      Or (And (Image all_black, Image (disk 5 5 5)), Image (disk 5 5 5))))
    6
    6
```

Correct value `true` 1 pt

Computing `render (Rem (Image (disk 5 5 5), Image (square 5 5 5))) 7 6`

Correct value `false` 1 pt

```
Computing
  render
    (And (And (Image (square 5 5 5), Image all_black),
      And (Image all_black, And (Image checkers, Image (disk 5 5 5)))))
    6
    7
```

Correct value `true` 1 pt

Exercise 3: `display_blend` Completed, 100 pts

Found `display_blend` with compatible type.

Computing `display_blend 10 10 (Image checkers)`

Expected output 5 pts

```
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
```

Computing
```
display_blend
  10
  10
  (Or
    (Rem (Rem (Image (square 5 5 5), Image all_black),
      And (Image (disk 5 5 5), Image all_black)),
    Or (Image (square 5 5 5), Image (disk 5 5 5))))
```
Expected output 5 pts
```
      #
   #######
  #########
  #########
  #########
 ###########
  #########
  #########
  #########
   #######
      #
```
Computing
```
display_blend
  10
  10
  (And (Image (disk 5 5 5),
    And (And (Image checkers, Image (square 5 5 5)), Image (disk 5 5 5))))
```
Expected output 5 pts
```
    #  ##
     ##
     ##
    #  ##
    #  ##
```
Computing
```
display_blend
  10
  10
  (And (And (Image (square 5 5 5), Image checkers), Image all_black))
```
Expected output 5 pts
```
    #  ##
     ##
     ##
    #  ##
    #  ##
```
Computing
```
display_blend
  10
  10
  (Rem (Rem (Image all_black, And (Image checkers, Image (disk 5 5 5))),
    Image (disk 5 5 5)))
```
Expected output 5 pts
```
#####  #####
##        ##
#          #
#          #
#          #

#          #
#          #
#          #
##        ##
#####  #####
```
Computing `display_blend 10 10 (Image (square 5 5 5))`

Expected output 5 pts
```
    #####
    #####
    #####
    #####
    #####
```

Computing `display_blend 10 10 (Image checkers)`

Expected output 5 pts

```
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
```

Computing `display_blend 10 10 (And (Image (disk 5 5 5), Image (disk 5 5 5)))`

Expected output 5 pts

```
        #
     #######
    #########
    #########
    #########
   ###########
    #########
    #########
    #########
     #######
        #
```

Computing `display_blend 10 10 (Image (square 5 5 5))`

Expected output 5 pts

```
     #####
     #####
     #####
     #####
     #####
```

Computing

```
  display_blend
    10
    10
    (And (And (Image (disk 5 5 5), And (Image checkers, Image (square 5 5 5))),
      Image (square 5 5 5)))
```

Expected output 5 pts

```
     #  ##
      ##
      ##
     #  ##
     #  ##
```

Computing `display_blend 10 10 (Image checkers)`

Expected output 5 pts

```
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ##  ##  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
```

Computing

```
  display_blend
    10
    10
    (Or
      (And (Rem (Image (disk 5 5 5), Image checkers),
        Rem (Image (disk 5 5 5), Image all_black)),
      Image (square 5 5 5)))
```

Expected output 5 pts

```
     #####
     #####
```

```
Computing
  display_blend
    10
    10
    (Or
      (Or (And (Image checkers, Image (disk 5 5 5)),
        Or (Image checkers, Image (square 5 5 5))),
      Rem (Image (square 5 5 5), And (Image all_black, Image checkers))))
```

Expected output 5 pts

```
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ######  #
  ## #######
  ## #######
    ######  #
    ######  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
```

```
Computing
  display_blend
    10
    10
    (Or
      (Or (And (Image checkers, Image all_black),
        And (Image all_black, Image (square 5 5 5))),
      And (Image (square 5 5 5), Or (Image all_black, Image (disk 5 5 5)))))
```

Expected output 5 pts

```
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
    ######  #
  ## #######
  ## #######
    ######  #
    ######  #
  ##  ##  ##
  ##  ##  ##
    ##  ##  #
```

```
Computing
  display_blend
    10
    10
    (Or (Rem (Image checkers, Image (disk 5 5 5)),
      Rem (Or (Image (square 5 5 5), Image (square 5 5 5)),
       Rem (Image checkers, Image (disk 5 5 5)))))
```

Expected output 5 pts

```
  ##  #   ##
  ##       #
            #
    #####  #
  #  #####
    #####
    #####  #
    #####  #
  #
  ##       #
    ##  ##  #
```

Computing `display_blend 10 10 (And (Image all_black, Image (disk 5 5 5)))`

Expected output 5 pts

```
       #
    #######
   #########
   #########
   #########
  ###########
   #########
   #########
   #########
    #######
       #
```

```
Computing
  display_blend
    10
    10
    (Rem (And (Image checkers, Image all_black),
      Or (Rem (Image all_black, Image (square 5 5 5)),
       Rem (Image (disk 5 5 5), Image checkers))))
```

Expected output 5 pts

```
     "  ""
      ##
      ##
     #  ##
     #  ##
```

Computing

```
  display_blend
    10
    10
    (Or (Rem (Or (Image all_black, Image (square 5 5 5)), Image (disk 5 5 5)),
      Image (square 5 5 5)))
```

Expected output 5 pts

```
  ##### #####
  ##       ##
  #         #
  #  #####  #
  #  #####  #
     #####
  #  #####  #
  #  #####  #
  #         #
  ##       ##
  ##### #####
```

Computing `display_blend 10 10 (Image (disk 5 5 5))`

Expected output 5 pts

```
        #
     #######
    #########
    #########
    #########
   ###########
    #########
    #########
    #########
     #######
        #
```

Computing

```
  display_blend
    10
    10
    (Or (Or (Image (disk 5 5 5), Image (square 5 5 5)), Image checkers))
```

Expected output 5 pts

```
  ##  ##  ##
  ##########
   ##########
   ##########
  ##########
  ###########
   ##########
   ##########
  ##########
  ##########
    ## ###  #
```