# Consignes pour le DM4

Pour le devoir 4, nous avons besoin d'afficher des images au format `png`. Il faut d'abord installer le module `camlimages` :

```
$  opam install graphics camlimages
```

On répond ensuite `Y` aux questions posées. L'installation se fait à priori sans problème.

### Rendu

Ce devoir est à réaliser seul ou en duo. Comme d'habitude l'archive `jean_dupont_marie_durant.zip` doit être déposée sur Cahier de Prépa avant le mardi 4 janvier minuit.

Votre fichier s'appelle `dm4.ml`. La commande de compilation est un peu longue (voir plus bas en 1) car il y a beaucoup de modules à importer. Une bonne idée serait de faire un `Makefile`.

En commentaire au début de `dm4.ml`, indiquer le.s nom.s de.s aut.eur.rice.s et l'adresse mail.

### Archive

Une archive est en ligne pour ce devoir. On extrait son contenu dans le répertoire courant : celui où vous compilerez vos codes.

Dans l'archive, outre ce fichier `pdf` de consignes, on trouve le sujet `dm4.pdf` proprement dit et trois fichiers images au format `png` : `flamantsNB.png, guepiersNB.png` et `rueNB.png`. Ce sont ces images que nous voulons compresser suivant différentes techniques.

## 1 Prise en main

### 1.1 Charger et afficher une image avec OCaml

Une fois l'achive extraite, le fichier `flamantsNB.png` est dans le répertoire courant. Dans le fichier `test.ml`, écrire le code suivant :

```
1  open Images
2
3  let () = Graphics.open_graph "";;
4
5  let img = Png.load "flamantsNB.png" [];;
6  let g = Graphic_image.of_image img;;
7
8  Graphics.draw_image g 0 0;;
9
10 Unix.sleep 5;; (*attendre 5 seconde et quitter*)
```

Compiler avec

```
$ ocamlfind ocamlc -o test -package graphics -package unix \
-package camlimages.png -package camlimages.graphics \
-linkpkg test.ml
```

Les antislash `\`, indiquent au terminal qu'on passe à la ligne sans avoir terminé l'écriture de la commande. Après cet antislash et un retour chariot, un chevron `>` apparaît sur la console et on peut écrire le reste de la commande à la suite. Si vous n'arrivez pas à les utiliser, écrivez toute la commande ci-dessus sur une seule ligne sans antislash.

Il suffit ensuite de lancer le programme :

```
$ ./test
```

Une fenêtre contenant l'image s'affiche en haut à gauche de votre écran (le pixel (0,0) indiqué par `Graphics.draw_image g 0 0`. Cet affichage dure 5 secondes.

*Remarque.* Si vraiment, vous n'y arrivez pas, vous pouvez toujours afficher l'image sans passer par `OCaml` avec

```
$ sudo apt-get install feh
$  feh flamantsNB.png
```

## 1.2 Affichage des transformations

Dans l'archive, on trouve le fichier `debut.ml` qui contient le début du code de votre devoir. Il contient notamment : :

- Un type `img` qui est un simple alias pour les matrices d'entiers :

```
type img = int array array;;
```

  Les entiers sont censés prendre des valeurs entre 0 et 255 [1]
  Ce type a vocation à représenter les images en niveau de gris dans le DM4. Un pixel blanc vaut 255, un noir 0. Par la suite nous dirons improprement *image* pour désigner une telle matrice.

- Une fontion `dim : img -> int * int` qui prend en paramètre une image en niveau de gris et retourne ses dimensions $w, h$ (nombre de colonnes et de lignes).
  Une image a **un nombre de ligne** $h$ **et un nombre de colonne** $w$. Cette convention est **valable** pour tout le devoir. En particulier, les complexités doivent toujours être exprimées en fonction de ces deux grandeurs au moins.

- Une fonction

```
(*valeur absolue d'un flottant*)
let abs x = if x > 0.0 then x else -. x;;
```

  qui fait ce qu'elle annonce.

1. Mon OCaml est un peu ancien, mais sur des versions plus récentes il existe un module `Unsigned . UInt8` pour implémenter le type `uint8_t` du C.

- Une fonction `load_matrix string -> img` de chargement d'une photo dont le nom est donné. Elle ouvre le fichier indiqué en paramètre et renvoie une image.
  Il n'est pas nécessaire de comprendre le code de cette fonction.
  C'est uniquement sur cette matrice retournée que vous effectuerez vos transformations.
- Une fonction `to_graphics img -> Graphics.image` qui transforme une image en niveau de gris en un objet `Graphics.image` exploitable par le module `Graphics` (notamment pour l'afficher). Ce module ne sait que manipuler des pixels au format RVB (Rouge, Vert, Bleu). C'est la raison pour laquelle nous tranformons chaque coefficient $x$ de la matrice en niveau de gris en un pixel dont tous les champs valent $x$ [2].
  Il n'est pas nécessaire de comprendre le code de cette fonction.
- Une fonction `display : string -> (img -> img) -> unit` qui prend en paramètres une image en niveau de gris et une fonction `f` de transformation d'image.
  La fonction `display` applique `f` à l'image ; ce qui donne une image transformée. Puis, l'image transformée est affichée durant 5 secondes dans une fenêtre en haut à gauche de l'écran.
  Il n'est pas nécessaire de comprendre le code de cette fonction.
- Une fonction `negative : img -> img` qui prend en paramètre une image et retourne le négatif de cette image.
  Cette fonction est donnée à titre d'exemple des transformations que vous devrez effectuer. Il est utile d'en comprendre le code.
- Une déclaration de type `unit` est donnée à titre d'exemple en fin de fichier.

```
let () =
  display  Sys.argv.(1) negative;;
```

Elle lance le calcul du négatif de la photo dont le nom est donné en premier paramètre du programme. L'image s'affiche dans une fenêtre durant 5 secondes. Pour vos tests, il vous suffira de remplacer `negative` par vos différentes fonctions.

Si vous voulez afficher une image sans modification :

```
let () =
  display  Sys.argv.(1) (fun x -> x);;
```

Pour utiliser ce début de code :

1. On compile avec :

```
$ ocamlfind ocamlc -o debut -package graphics -package unix\
  -package camlimages.png -package camlimages.graphics\
  -linkpkg debut.ml
```

2. On appelle le programme en lui donnant en paramètre le nom d'un fichier en niveau de gris :

```
$  ./debut flamantsNB.png
```

Une image s'affiche quelques secondes.

2. Pour faire un pixel gris au format RVB, il faut mettre tous les champs à la même valeur. Plus cette valeur est proche de 255, plus le gris est clair.

## Complexité

Dans le devoir, il est demandé pour certaine question de donner une estimation de la complexité. En commentaire au dessus du code de la fonction, on écrit :

```
(*Q1 O(h+w^2)*)
let f (mat:img):img = ...
```

Pour toute fonction dont il est demandé la complexité, c'est de cette manière **(avec le numéro de question et une réponse en grand O)** que vous communiquerez le résultat de votre estimation (on rappelle que celle-ci est fonction de $w$ et $h$).

Pour d'autres question, on demande de respecter une complexité (par exemple $O(nhw)$ pour la compression Seam carving). Il faut donc bien faire attention à respecter cette consigne. Si votre complexité est plus grande qu'attendue, vous vous en rendrez compte par vous-même : la compression de certaines images prendra alors beauuuuucoup de temps !

Souvenons-nous qu'appeler dans une boucle une fonction dont le code utilise lui même une boucle multiplie en général les complexités. Dans certains cas, on ne peut pas faire autrement. Mais il faut toujours se poser la question : cette fonction que j'appelle dans la boucle ne pourrait-elle pas être appelée avant ou après la boucle pour un résultat identique ?

## Fonctions autorisées

La plupart des fonctions demandées sont en double (voire triple) boucle imbriquées.

Les Kadors (au moins 15 de moyenne en info) à l'aise avec ces notions peuvent toutefois utiliser toutes les fonction des modules `Array` et `Hashtbl` ; en particulier `Array.map`, `Array.iter`, `Array.mapi` et `Array.iteri` [3] dont je me suis souvent servies. Attention, ces fonctions sont élégantes mais cachent une boucle. Si vous les utilisez, il faudra en tenir compte dans votre estimation de la complexité.

## Explications

Certaines questions amènent des réponses en Français. Lorsque c'est le cas, la réponse est à donner en commentaire dans le code.

```
(*Q9c L'image obtenue après réduction des 200 meilleures
lignes dans la photo des flamants est mauvaise à cause
de l'âge du capitaine. *)

(*Q14 le tableau des énergies minimales est
20 10 30 45
 3  4  0  5
 0 12 29 44
 *)
```

3. `map` et `mapi` créent un nouveau tableau, tandis que `iter` et `iteri` font des effets de bords