T.P. 02
Programmation Impérative Bis

■ ■ ■

Dans cette feuille d'exercice nous allons dessiner une fractale assez célèbre. L'ensemble de Mandelbrot, puisque c'est son nom, est un ensemble de points du plan complexe faisant converger une certaine suite définie par récurrence.

I - Suites récurrentes complexes

Nous représenterons les nombres complexes par le type `float * float` (un flottant pour la partie réelle et un pour la partie imaginaire).

Exercice 1. (Opérations sur les complexes)
Écrire des fonctions d'addition et de multiplication complexe, ainsi qu'une fonction calculant le carré de la norme d'un complexe.

```
add    : float * float -> float * float -> float * float
mul    : float * float -> float * float -> float * float
norme2 : float * float -> float
```

Étant donné un nombre complexe c, on considère la suite complexe définie par les équations suivantes :

$$\begin{cases} Z_0 &= 0 \\ Z_{n+1} &= Z_n^2 + c \end{cases}$$

Officiellement, l'ensemble de Mandelbrot est l'ensemble des complexes c tels que la suite précédente ne tend pas vers l'infini (en norme). Nous prendrons une approximation de cette définition, dans laquelle nous regarderons seulement si le carré de la norme dépasse une certanie valeur $R = 4$ lors des $N = 50$ premières itérations.

Nous définissons donc

```
limite_iterations = 50
limite_norme2     = 4
```

Exercice 2. (Estimation de la divergence)
Écrire une fonction `divergence : float * float -> int` qui prend en entrée un nombre complexe, et retourne :

$$\begin{cases} n \text{ si la limite } R \text{ a été dépassée à la } n^{\text{ème}} \text{ itération } (n < N), \\ N \text{ sinon.} \end{cases}$$

II - Affichage

À chaque nombre complexe c nous associerons une couleur (en fait un « niveau de gris »), qui sera d'autant plus foncée que la suite récurrente aura divergé vite. Pour définir une couleur, nous utiliserons la fonction `rgb` de Caml, qui prend en entrée trois entiers entre 0 et 255 et retourne une couleur. Les trois entrées correspondent aux quantités de rouge, de vert et de bleu composant la couleur finale. En particulier `rgb 0 0 0` donnera du noir et `255 255 255` du blanc.

Exercice 3. (Normalisation de la couleur)
Écrire une fonction `gris` qui prend en entrée un nombre d'itérations n et renvoie un niveau de gris proportionnel au rapport $\frac{n}{N}$.

```
gris : int -> color
```

Pour afficher le résultat, il va aussi falloir faire un zoom, afin que les coordonnées entières de l'écran suffisent à afficher quelque chose d'intéressant. Voici la formule proposée : pour un affichage de dimensions $taille_x$ et $taille_y$, le pixel de coordonnées (x, y) affichera la couleur correspondant au nombre complexe :

$$(\frac{2x}{taille_x} - 1.5) \quad + \quad i\,(\frac{2y}{taille_y} - 1)$$

Exercice 4. (Normalisation des coordonnées)
Écrire une fonction `point` qui prend en entrée une paire de dimensions `(taille_x, taille_y)` et une paire de coordonnées `(x, y)` et qui retourne le nombre complexe correspondant.

```
point : int * int -> int * int -> float * float
```

Il ne reste plus qu'à tout mettre ensemble !

Exercice 5. (Dessin)
En complétant le squelette donné ci-dessous, écrire une fonction `mandelbrot : int -> int -> ()` qui prend en entrée des dimensions `taille_x` et `taille_y` et affiche l'ensemble de Mandelbrot dans une fenêtre de taille correspondante.

Fonctions utiles :

`plot : int -> int -> ()`	prend des coordonnées et affiche un point.
`set_color : color -> ()`	détermine la prochaine couleur utilisée.

Squelette à compléter :

```
let mandelbrot taille_x taille_y =
  open_graph (printf__sprintf " %dx%d" (taille_x+30) (taille_y+60));

  ... (* Ça c'est pour vous. *)

  let _ = read_key () in
  close_graph ()
```