TROIS FRACTALES

TP

```
from tkinter     import Tk, Canvas
from PIL.Image   import new
from PIL.ImageTk import PhotoImage
```

```
WIDTH = 600 ; HEIGHT = 600
```

```
def Afficher(Dessin) :

    F = Tk() ; F.title("Mon dessin")
    D = Canvas(F, width = WIDTH, height = HEIGHT)
    D.pack()

    PhImg_Dessin = PhotoImage(Dessin)
    D.create_image((1 + WIDTH // 2 , 1 + HEIGHT // 2), image=PhImg_Dessin)

    F.mainloop()
```

```
def CréerDessin() :
    return new("RGB", (WIDTH , HEIGHT), "white")
```

```
x_min = -3
x_max =  3
y_min = -3
y_max =  3
```

```
def PixelVersPoint(px, py) :
    x = x_min + px / (WIDTH - 1) * (x_max - x_min)
    y = y_max - py / (HEIGHT - 1) * (y_max - y_min)
    return (x, y)
def PointVersPixel(x, y) :
    px = int( (x - x_min)/(x_max - x_min) * (WIDTH - 1) )
    py = int( (y_max - y)/(y_max - y_min) * (HEIGHT - 1) )
    return (px, py)
```

Partie I — Mandelbrot & cie

Soit c un nombre complexe. On considère les suites $(z_n)_{n\geqslant 0}$ définies par la relation de récurrence

$$z_{n+1} = z_n^2 + c.$$

On appelle *ensemble de Julia rempli* l'ensemble

$$\mathscr{R}(c) = \{z_0 \in \mathbf{C} \mid |z_n| \leqslant 2 \text{ pour tout } n \geqslant 0\}$$

des valeurs initiales pour lesquelles la suite reste bornée, et *ensemble de Mandelbrot* les valeurs de c pour lesquelles l'ensemble de Julia rempli contient zéro, c'est-à-dire

$$\mathscr{M} = \{c \in \mathbf{C} \mid 0 \in \mathscr{R}(c)\}.$$

On va essayer de dessiner ces ensembles. Et comme on ne peut pas calculer *tous* les termes d'une suite, on définira un indice maximal

```
N_MAX = 50
```

au-dela duquel on n'arrêtera les calculs.

Exercice 1 — Tester le code ci-dessous. Que faut-il changer pour que le fond soit noir ? Pour déplacer le disque ? Pour qu'il soit vert ?

```
def FonctionModèle() :

    Dessin = CréerDessin()

    for px in range(0, WIDTH) :
        for py in range(0, HEIGHT) :

            (x, y) = PixelVersPoint(px, py)
            if x ** 2 + y ** 2 <= 1 :
                C = (0, 0, 255)
            else :
                C = (255, 255, 255)
            Dessin.putpixel( (px, py), C )

    return Dessin
```

Exercice 2 — Écrire un programme `IndiceSortie(`c`, z_0 = 0)` qui renvoie le plus petit indice $n \geqslant 0$ pour lequel on a $|z_n| > 2$. On renverra `N_MAX` si cet indice n'existe pas, ou bien s'il est plus grand que `N_MAX`.

Exercice 3 — Dégradés.

1) Proposer une fonction D_x qui à tout élément $\lambda \in [0\,;\,1]$ associe un entier entre 0 et x, allant progressivement de 0 (pour $\lambda = 0$) à x (pour $\lambda = 1$).

2) En déduire un programme `Dégradé(`n`, C)` qui à un triplet $\mathrm{C} = (r, g, b)$ associe la couleur

$$(\mathrm{D}_r(n/\texttt{N\_MAX}), \mathrm{D}_g(n/\texttt{N\_MAX}), \mathrm{D}_b(n/\texttt{N\_MAX})).$$

Exercice 4

1) Écrire le programme `Mandelbrot()` qui dessine l'ensemble de Mandelbrot $\mathscr{M}$.

2) Écrire le programme `Julia(`c`)` qui dessine l'ensemble de $\mathscr{R}(c)$.

Partie II — Objets de Sierpinski

Soit N un entier strictement positif. On se donne une liste `V` de couples v_k, de longueur K, représentant des vecteurs du plan, ainsi qu'un réel $\lambda \in]0\,;\,1[$.

À chaque réel $t \in [0\,;\,1[$ on associe le point (x, y) du plan défini par

$$\mathrm{P_V}(t) = \begin{pmatrix} x \\ y \end{pmatrix} = \sum_{n=1}^{\mathrm{N}} \lambda^{n-1} \times v_{t_n},$$

où $t = \overline{0{,}t_1t_2t_3\ldots t_\mathrm{N}}$ est l'écriture en base K de t.

Exercice 5

1) Construire la liste `Fanion` formée des vecteurs d'affixes $e^{i\pi/2}$, $e^{7i\pi/6}$ et $e^{-\pi/6}$.

2) Construire la liste `Moquette` formée des vecteurs d'affixes 1, $\sqrt{2}e^{i\pi/4}$, i, $\sqrt{2}e^{3i\pi/4}$, -1, $\sqrt{2}e^{-3i\pi/4}$, $-i$ et $\sqrt{2}e^{-i\pi/4}$.

Les objets de Sierpinski peuvent être dessinés de la manière suivante : on tire au hasard un grand nombre M de réels t dans $[0\,;1[$, et pour chacun d'eux on peint (en noir par exemple) le pixel qui correspond à $\mathrm{P_V}(t)$.

Exercice 6

1) Dessiner le *fanion* de Sierpinski, en utilisant le rapport $\lambda = 1/2$.

2) Dessiner la *moquette* de Sierpinski, en utilisant le rapport $\lambda = 1/3$.

Partie III — Fractales de Newton

Dans toute cette partie on fixe une fonction polynomiale P, de degré au moins deux. Pour tester on commencera avec celle-ci.

```
def P(z) :
    return z ** 3 - 1
```

Exercice 7 — Écrire un programme `NombreDérivé(P, z)` qui calcule et renvoie une valeur approchée de $\mathrm{P}'(z)$.

Rappelons la méthode de Newton : on part d'un nombre complexe z_0 et on construit une suite avec la relation de récurrence

$$z_{n+1} = z_n - \frac{\mathrm{P}(z_n)}{\mathrm{P}'(z_n)}.$$

Exercice 8

1) On suppose que la suite $(z_n)_{n\geqslant 0}$ est définie pour tous les indices n, et qu'elle converge vers $\ell \in \mathbf{C}$. Prouver que $\mathrm{P}(\ell) = 0$.

2) Écrire le programme `Newton(P, z0, ε = 1e-10)` qui applique l'itération de Newton jusqu'à avoir $|z_{n+1} - z_n| \leqslant \varepsilon$. On renverra les valeurs de z_{n+1} et de n, et comme dans la première partie, on s'arrête au pire à $n =$ `N_MAX`.

Exercice 9 — Écrire un programme `AjouterPeutÊtre(L, z)` qui ajoute un nombre complexe à une liste, s'il ne s'y trouve pas déjà. On prendra garde au fait que les nombres complexes sont des valeurs *approchées*, et qu'on ne peut donc pas brutalement utiliser `==`.

Exercice 10

1) On suppose que la suite $(z_n)_{n\geqslant 0}$ est définie pour tous les indices n, et qu'elle converge vers $\ell \in \mathbf{C}$. Prouver que $\mathrm{P}(\ell) = 0$.

2) Écrire le programme `Newton(P, z0, ε = 1e-10)` qui applique l'itération de Newton jusqu'à avoir $|z_{n+1} - z_n| \leqslant \varepsilon$. On renverra les valeurs de z_{n+1} et de n, et comme dans la première partie, on s'arrête au pire à $n =$ `N_MAX`.

Le fractale de Newton associé à P s'obtient ainsi. On commence par associer, arbitrairement, une couleur à chaque racine de P. Puis, pour chaque pixel du plan, on considère l'affixe $z_0 = x + iy$ qui lui est associée, on cherche la limite ℓ de la suite $(z_n)_{n\geqslant 0}$, et on peint le pixel de départ de la couleur associée à la racine ℓ.

Pour que ce soit plus joli, on fait un dégradé en fonction de la « vitesse » à laquelle on a convergé vers ℓ, c'est-à-dire le nombre n d'itérations effectuées.

Exercice 11

1) Écrire un programme qui étant donné un entier d (représentant le degré du polynôme P construit une liste de d couleurs choisies aléatoirement.

2) En déduire le programme `FractaleDeNewton()` qui dessine le fractale associé à P.