Le but de ce TP est de se familiariser avec le concept de *récursivité* en dessinant des fractales avec le module `turtle` de Python. La récursivité réfère au fait qu'une fonction fasse référence à elle-même. Par exemple, voici en Python une définition récursive de la factorielle

```
def factorial(n):
    if n==0:
        return 1
    else:
        return n * factorial(n-1)
```

19.1 Introduction à `turtle`

Le module `turtle` de Python permet de contrôler une "tortue" qui, en se déplaçant sur une feuille blanche, laisse un trait de crayon. On peut donc dessiner en contrôlant les mouvements de la tortue. Les commandes les plus utiles sont

- `forward(step)` : avance la tortue d'une distance `step`
- `backward(step)` : recule la tortue d'une distance `step`
- `left(angle)` : tourne la tortue de `angle` degrés vers la gauche
- `right(angle)` : tourne la tortue de `angle` degrés vers la droite

On peut également contrôler la vitesse de déplacement de la tortue (utile pour la ralentir lorsque vous débuggez, ou pour l'accélérer lorsque le dessin prend du temps à être construit).

- `speed(0)` : vitesse la plus rapide de la tortue
- `speed(1)` : vitesse la plus lente de la tortue
- `speed(5)` : vitesse moyenne
- `speed(10)` : vitesse rapide

Par ailleurs, on peut remplir ("*fill*") l'intérieur de la courbe dessinée par la tortue en encadrant le mouvement par

```
begin_fill()
... # la tortue fait des mouvements
end_fill()
```

Attention, à la fin de votre code, il faut absolument appeler la fonction `done` du module `turtle`.

```
... # mouvements de tortue
done()
```

19.2 4 challenges `turtle`

Voici 4 problèmes de difficulté croissante. On essaiera de les résoudre avec le code le plus concis possible.

19.2.1 Challenge 1

Construire avec `turtle` la courbe figure 19.1.

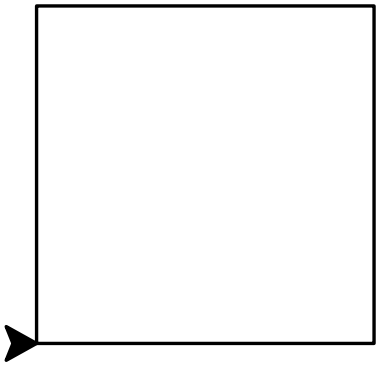


FIGURE 19.1 – Un carré

19.2.2 Challenge 2

Construire avec `turtle` la courbe figure 19.2. On pourra écrire une fonction `polygone(n)` qui construit un polygône régulier à n côtés.

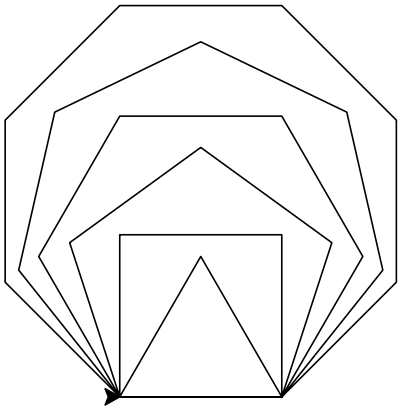


FIGURE 19.2 – Des polygônes réguliers

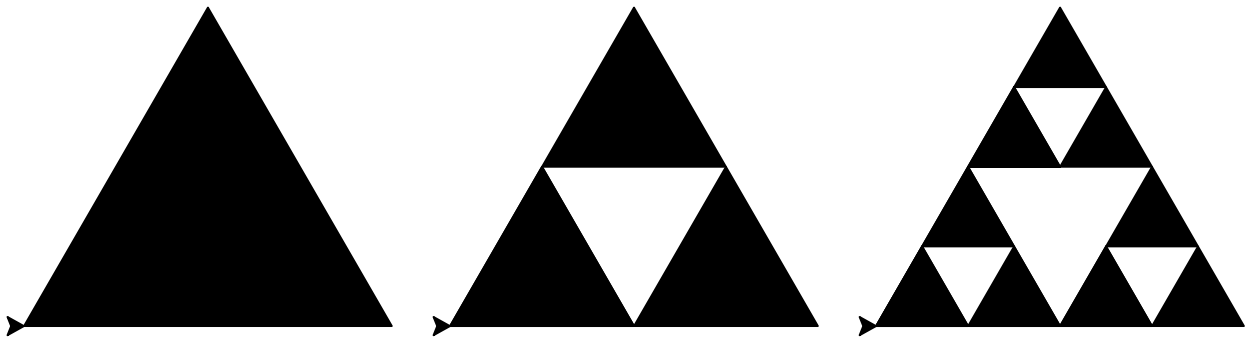
19.2.3 Challenge 3

Ce challenge propose de construire un triangle de Sierpinsky, qui est une fractale formée de triangles dans des triangles dans des triangles etc. Construire avec `turtle` le triangle de Sierpinsky à l'étape 5 Figure 19.3d. Il faudra écrire de façon récursive une fonction `sierpinsky(n)` qui construit le triangle de Sierpinsky à l'étape n, en remarquant que les 3 sous-triangles de l'étape n sont des triangles de l'étape $n-1$.

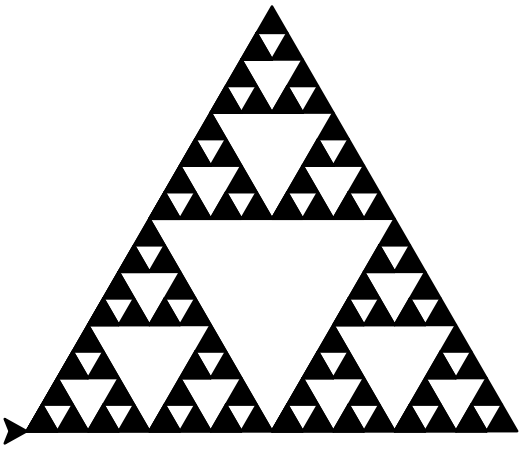
19.2.4 Challenge 4

Ce challenge propose de construire la courbe de Hilbert, qui est une courbe fractale remplissant le carré unitaire. Construire avec `turtle` la courbe de Hilbert à l'étape 5 Figure 19.4d. Il faudra écrire de façon récursive une fonction `hilbert(n)` qui construit la courbe de Hilbert à l'étape n.

Attention, les occurrences fractales à l'étape n de la courbe de Hilbert à l'étape $n-1$ sont parfois symmétrisées. On pourra avoir besoin d'écrire deux fonctions *mutuellement récursives*, `hilbertDirect(n)` et `hilbertIndirect(n)` qui construisent les courbes de Hilbert en tournant dans le sens direct et indirect, respectivement (voir figure 19.5). Cependant, on peut rendre le code encore plus concis...

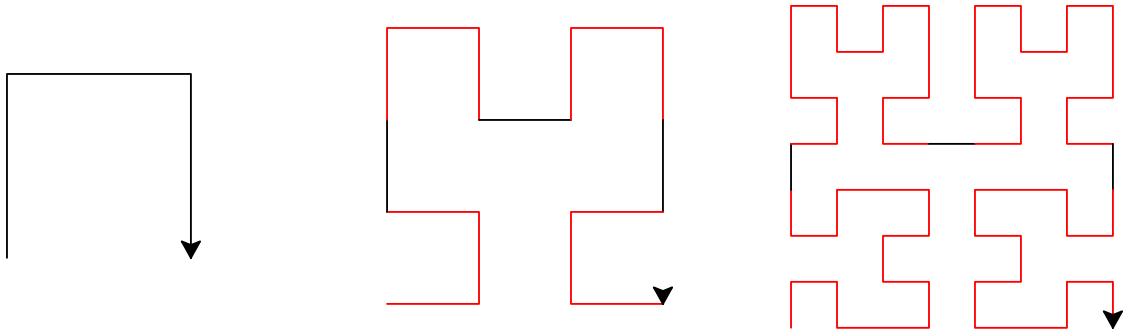


(a) Étape 1 (b) Étape 2 (c) Étape 3

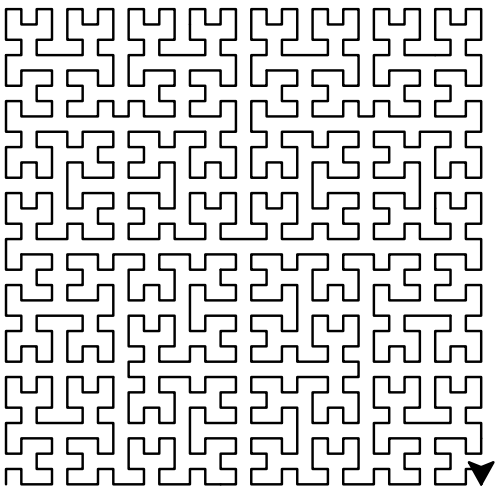


(d) Étape 5

FIGURE 19.3 – Triangle de Sierpinsky.

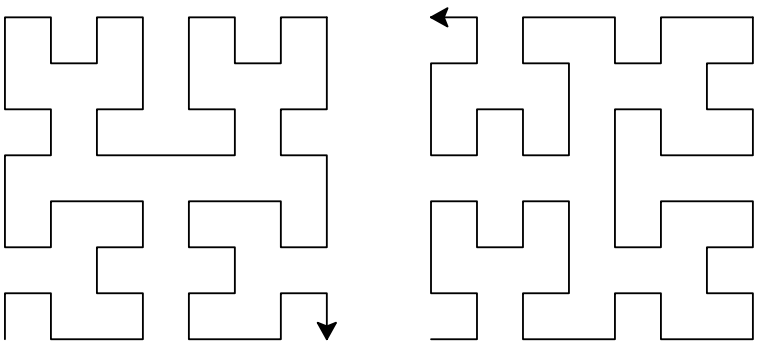


(a) Étape 1 (b) Étape 2 (c) Étape 3



(d) Étape 5

FIGURE 19.4 – Courbe de Hilbert. On met en évidence le caractère fractal de la courbe aux étapes 2 et 3, en coloriant les occurrences des étapes 1 et 2, respectivement.



(a) Courbe de Hilbert dans le sens direct (b) Courbe de Hilbert dans le sens indirect

FIGURE 19.5 – Courbes de Hilbert en "tournant dans des sens différents"