Informatique
TP5 : Interface graphiques et tracés de fractales
CPP 1A

Djamel Aouane, Frederic Devernay, Matthieu Moy

Mars - avril 2015

Ce TP est dédié à la construction de courbes fractales. La notion de *récursivité* peut être illustrée géométriquement par des *fractales*. Ce TP débute par une présentation de fonctions graphiques de Python. Il vous est ensuite demandé de construire un certain nombre de courbes "fractales".

1 Le module `turtle` de Python

Nous allons utiliser l'outil `turtle` afin de réaliser ce TP. La fenêtre graphique Turtle de Python est assimilable à un ensemble de points d'un plan. Chaque point est désigné par deux coordonnées entières sur le plan. L'origine (coordonnées $(0, 0)$) est située au centre de la fenêtre. La taille de la fenêtre par défaut est de $(400, 300)$.

1.1 Les fonctions graphiques de base

Le principe de `turtle` est simple : la tortue est le triangle affiché au milieu de l'écran. On peut faire avancer, reculer, et tourner la tortue. Par défaut, la tortue trace un trait derrière elle quand elle se déplace.

Exercice 1 *Entrez ceci dans l'interprète Python :*

```
>>> from turtle import *
>>> forward(100)
>>> right(120)
>>> forward(100)
>>> right(120)
>>> forward(100)
```

Vous devriez voir apparaître un triangle.

La première ligne dit à Python que l'on va utiliser `turtle`. Les suivantes sont des instructions qu'on donne à la tortue. Bien sûr, on peut combiner ces instructions avec les constructions Python classiques. Par exemple, dessiner un triangle comme ci-dessus est aussi simple que :

```
1 from turtle import *
2
3 for i in range(3):
4     forward(100)
5     right(120)
6
7 mainloop()
```

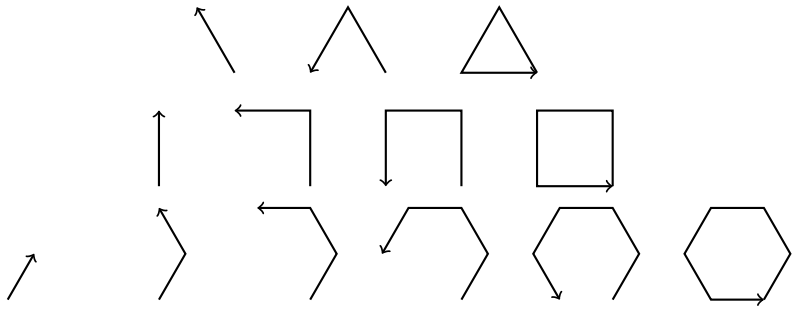


FIGURE 1 – Fonctionnement de *trace_polygone*

(La ligne `mainloop()` permet d'éviter que la fenêtre se ferme toute seule à la fin du programme)
Plus précisément, le TP nécessite l'utilisation des fonctions suivantes :
- `forward(`*`longueur`*`)` : fait avancer la tortue de *longueur* pixels.
- `left(`*`angle`*`)`, `right(`*`angle`*`)` : fait tourner la tortue sur elle même de *angle*, en degrés.
- `penup()` : permet de lever le crayon en vue de déplacement sans tracer.
- `pendown()` : pose le crayon, le prochain `forward()` tracera un trait.
- `goto(x, y)` ou `goto([x, y])` : sert à déplacer le curseur vers un point dont on précise les coordonnées.

Remarque : Pour des informations complémentaires sur le graphisme en Python (il existe d'autres fonctions), vous pouvez consulter le manuel (voir la page `https://docs.python.org/3.4/library/turtle.html`).

1.2 Tracé de quelques figures simples

Exercice 2 (Tracé d'un carré) *Modifiez le programme de tracé de triangle ci-dessus pour tracer un carré (il est conseillé de l'écrire dans un fichier, mais ce fichier ne* doit pas *s'appeler* `turtle.py`*).*

Exercice 3 (Tracé d'un carré avec une fonction) *Modifiez le programme de tracer de carré pour "emballer" le code dans une fonction. La longueur du côté est un paramètre de la fonction :*

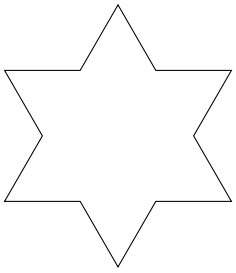
```
1 def trace_carre(longueur):
2     # à vous de compléter
```

Appelez cette fonction plusieurs fois avec plusieurs paramètres différents.

Exercice 4 *Pour généraliser l'exercice précédent, écrire une fonction* `trace_polygone(longueur, nb_cotes)` *qui trace un polygone à* `nb_cotes` *côtés.*

La figure 1 montre les étapes du tracé d'un triangle, d'un carré et d'un hexagone.

Exercice 5 (Tracé d'une étoile) *Écrire un programme Python dessinant une étoile à 6 branches, comme ceci :*



Indice : pour tracer une branche de l'étoile, il faut deux segments donc deux appels à `forward()` *(avec un appel à* `right()` *pour former le sommet de l'étoile). Avec une boucle, on évite de dessiner les 6 branches à la main.*

Exercice 6 *Que fait le programme suivant ? Essayez de deviner avant d'exécuter le programme.*

```
from turtle import *

for i in range(360):
    forward(1)
    right(1)

mainloop()
```

Exercice 7 *Que fait le programme suivant ? Essayez de deviner avant d'exécuter le programme.*

```
from turtle import *

for i in range(5):
    forward(200)
    right(360 * 2/5)

mainloop()
```

2 Une première courbe fractale : le flocon de von Koch

2.1 Les fractales

Une fractale est une sorte de courbe mathématique un peu complexe extrêmement riche en détails, et qui possède une propriété intéressante visuellement : lorsque l'on regarde des détails de petite taille, on retrouve des formes correspondant aux détails de plus grande taille (auto-similarité).

2.2 Le flocon de von Koch

La première courbe à tracer a été imaginée par le mathématicien suédois Niels Fabian Helge von Koch, afin de montrer que l'on pouvait tracer des courbes continues en tout point, mais dérivables en aucun.

Le principe est simple : on divise un segment initial en trois morceaux, et on construit un triangle équilatéral sans base au-dessus du morceau central. On réitère le processus n fois, n étant appelé l'ordre. Dans la figure suivante on voit les ordres 0, 1, 2 et 3 de cette fractale.

Si on trace trois fois cette figure, on obtient successivement un triangle, une étoile, puis un flocon de plus en plus complexe :

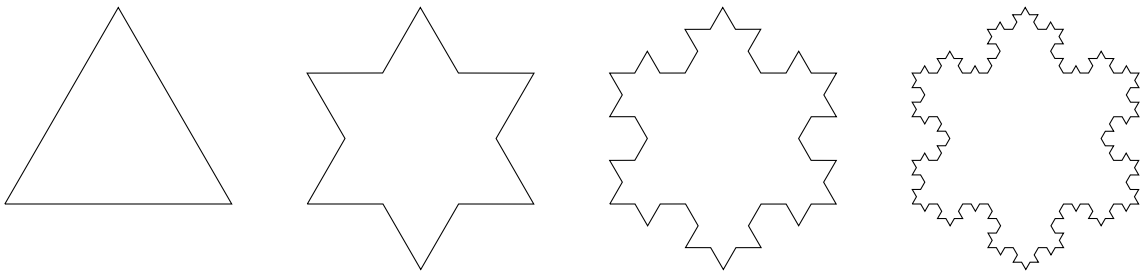


FIGURE 2 – Le Chou Romanesco : exemple classique de fractale naturelle

Le dessin de la fractale à l'ordre 0 est trivial, il suffit d'appeler `forward()` une fois.

À l'ordre 1, on va appeler quatre fois `forward()`, en appelant `left()` et `right()` pour changer la direction de la tortue entre chaque segments.

À l'ordre $n \geq 1$, on va appliquer le même principe qu'à l'ordre 1, mais en remplaçant les appels à `forward()` par des dessins de segments à l'ordre $n - 1$.

Une fractale ressemblant au flocon de Koch existe chez votre marchand de légumes : le Chou Romanesco (figure 2) est constitué de plusieurs pointes, chacune constituée de plusieurs pointes, chacune constituée de plusieurs pointes ... jusqu'à l'ordre 5 ou 6 !

Exercice 8 (Koch à l'ordre 1) *Écrivez une fonction* `koch_1(longueur)` *qui dessine un segment de Koch à l'ordre 1, de la longueur spécifiée. Les sous-segments de la figure sont de longueur* $\frac{longueur}{3}$*, et les angles sont de 60 ou 120 degrés.*

Appelez cette fonction depuis votre programme pour vérifier son fonctionnement.

Exercice 9 (Début de généralisation : ordre 0 ou 1) *Modifiez votre fonction pour lui faire prendre en paramètre l'ordre de la fractale. La fonction est maintenant* `koch(n, longueur)`*, ou n est l'ordre.*

Modifiez le corps de la fonction pour qu'elle gère correctement les cas $n = 0$ et $n = 1$ (le cas $n \geq 1$ viendra plus tard). Le code va ressembler à :

```
if n == 0:
    # Cas n == 0 (trivial en utilisant forward()).
else:
    # cas n == 1, comme ci-dessus.
```

Vérifiez que la fonction marche correctement pour ces deux valeurs.

Étonnement, il n'y a presque rien à changer pour généraliser notre fonction à une valeur quelconque de n. Pour l'instant, le cas $n = 1$ appelle 4 fois la fonction `forward`, qui correspond au cas $n = 0$. Nous avons vu que la fractale à l'ordre n devait utiliser la fractale à l'ordre $n - 1$ (c'est le cas puisque $0 = 1 - 1$). En remplaçant les appels à `forward` par des appels à `koch(n - 1, ...)`, on ne changera pas le comportement de notre fonction `koch` à l'ordre 1, mais on lui permettra de gérer correctement les ordres $n \geq 2$.

Exercice 10 (Koch à l'ordre n) *Modifiez la fonction Koch comme expliqué ci-dessus pour gérer les ordres $n \geq 2$. Testez votre fonction avec différentes valeurs de n (en pratique, on ne voit plus grand chose avec un ordre supérieur à 5 ou 6).*

Une manière de tester est d'utiliser le morceau de code suivant :

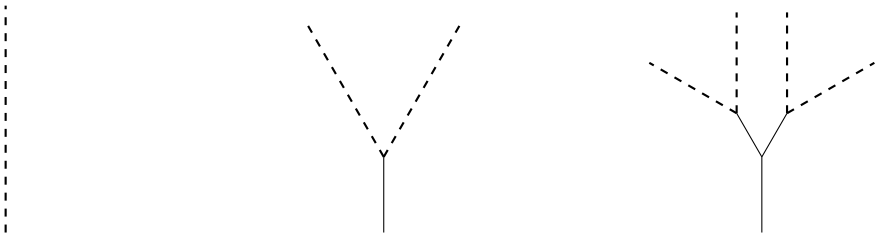
```
for i in range(10):
    penup()
    goto(0, 70 * i)
    pendown()
    koch(i, 300)
```

L'exécution peut être assez lente. Deux astuces pour l'accélérer :
- Appeler `speed(0)` avant de dessiner : ceci va régler la vitesse de la tortue au maximum.
- Appeler `hideturtle()` avant de dessiner, et `showturtle()` après. Le triangle représentant la tortue ne sera pas redessiné à chaque étape, on gagne beaucoup de temps.

3 Arbre fractal

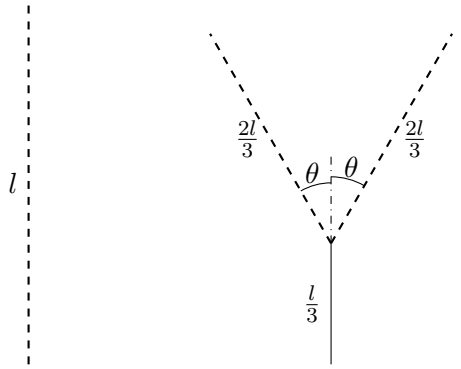
On peut appliquer le même principe pour construire d'autre fractales visuellement différentes, mais construites sur le même principe.

Par exemple, pour construire un arbre, on part d'un segment, et on applique la transformation suivante à chaque segment de la construction (on refait la transformation n fois pour obtenir un arbre d'ordre n) :



Les portions dessinées en pointillées sont celles sur lesquelles on va appliquer la transformation à l'ordre suivant (on les appellera les segments non-terminaux). Les portions dessinées en trait plein sont des segments qui ne seront pas transformés (on les appellera les segments terminaux).

Plus précisément, la transformation à appliquer est la suivante :

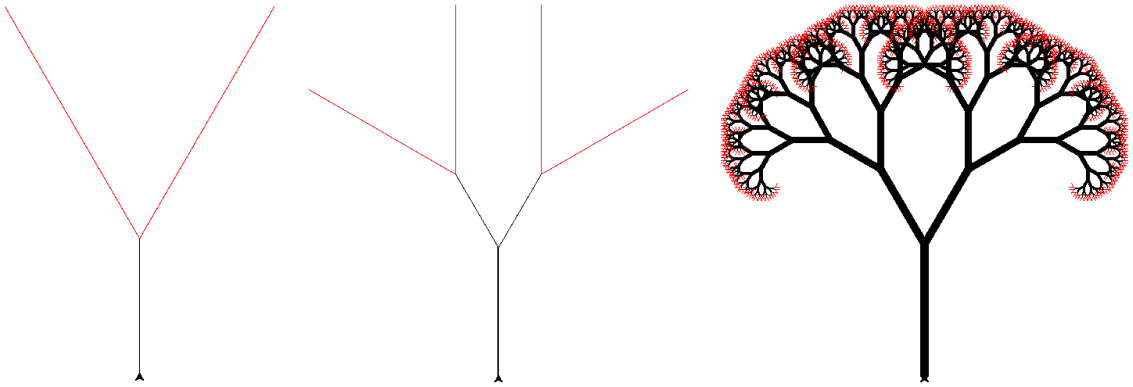


Pour transformer un segment non-terminal de longueur l, on trace un segment terminal de longueur $\frac{l}{3}$, puis deux segments non-terminaux de longueur $\frac{2l}{3}$ à un angle θ du premier segment. On peut choisir $\theta = 30$ degrés par exemple.

Exercice 11 (Arbre fractal) *Suivez la même démarche que pour le flocon de Koch pour écrire une fonction* `arbre(n, longueur)` *qui trace l'arbre, et repositionne la tortue à son point de départ.*
- *Dans un premier temps, gérer l'ordre 0 (trivial) et 1 (à base de* `forward()` *et sans appel récursif), et tester votre fonction. Vérifiez en particulier que la tortue est bien revenue à son point de départ à la fin de l'exécution de la fonction dans les deux cas ($n = 0$ et $n = 1$).*

– *Pour chaque tracé de segment non-terminal, remplacez l'appel à* `forward()` *par un appel récursif à* `arbre()`*, et vérifiez le comportement de votre fonction à différents ordres. Cette fois-ci, une profondeur 10 ou 11 donne un résultat assez joli.*

Si on veut un dessin plus joli, on peut jouer sur l'épaisseur des traits et les couleurs. Par exemple, si on choisit comme épaisseur de trait la valeur de `n` pour les valeurs de `n` non-nulles, et qu'on colorie en rouge les segments tracés pour `n == 0`, on obtient la figure 3.



(a) `arbre(1, 300)` (b) `arbre(2, 300)` (c) `arbre(11, 300)`

FIGURE 3 – Copie d'écran du résultat de l'exéction de `arbre()` pour différentes valeurs de `n`

Exercice 12 (Question subsidiaire ...) *Combien feuilles a un arbre à l'ordre n ? Combien de segments élémentaires a un segment de Koch à l'ordre n ?*

Pour ceux qui n'en ont jamais assez : sur le même principe, on peut dessiner la courbe du dragon, ou la courbe de Hilbert (qui permet de montrer que $\mathbb{R}$ et $\mathbb{R}^2$ ont le même cardinal). Wikipedia est votre ami ;-).

4 Solution des exercices

Exercices 2, 3 et 4 : tracer un carré :

```
from turtle import *

for i in range(4):
    forward(100)
    right(90)

def trace_carre(longueur):
    for i in range(4):
        forward(longueur)
        right(90)

def trace_polygone(longueur, nb_cote):
    for i in range(nb_cote):
        forward(longueur)
        right(360 / nb_cote)
```

Exercice 5 : étoile à 6 branches

```
from turtle import *

for i in range(6):
    forward(100)
    right(120)
    forward(100)
    left(60)

mainloop()
```

Exercice 6 : le code trace un polygone régulier à 360 côtés. Visuellement, c'est un cercle.
Exercice 7 : c'est une étoile à 5 branches.
Exercice 8 : Koch à l'ordre 1

```
def koch_1(longueur):
    forward(longueur/3)
    left(60)
    forward(longueur/3)
    right(120)
    forward(longueur/3)
    left(60)
    forward(longueur/3)
```

Exercice 10 : Koch à l'ordre n

```
from turtle import *

speed(0) # Pour accélérer la tortue

def koch(n, longueur):
    if n == 0:
        forward(longueur)
    else:
        koch(n - 1, longueur/3)
        left(60)
        koch(n - 1, longueur/3)
        right(120)
        koch(n - 1, longueur/3)
        left(60)
```

```
        koch(n - 1, longueur/3)

for i in range(10):
    penup()
    goto(0, 70 * i)
    pendown()
    koch(i, 300)

# Pour que la fenêtre ne se ferme pas tout de suite
mainloop()
```

Exercice 11 : Arbre fractal

```
from turtle import *

speed(0)

angle=30

def arbre(n, longueur):
    if n == 0:
        color("red")
        forward(longueur)
        backward(longueur)
        color("black")
    else:
        width(n)
        forward(longueur / 3)
        left(angle)
        arbre(n - 1, longueur / 3 * 2)
        right(2*angle)
        arbre(n - 1, longueur / 3 * 2)
        left(angle)
        backward(longueur / 3)

left(90)
hideturtle()
arbre(11, 500)
showturtle()

mainloop()
```

Exercice 12 : 2^n feuilles pour l'arbre, 4^n segents pour Koch. Ça croît vite (2048 feuilles pour notre arbre à $n = 11$, 4096 côté pour Koch à l'ordre 6) !