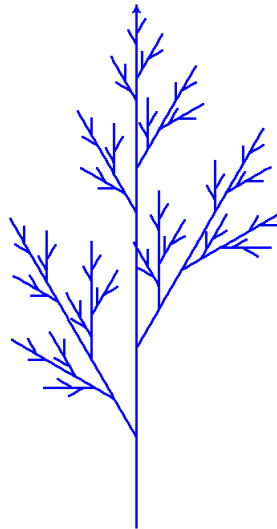


# L-système

Les L-systèmes offrent une façon très simple de coder des phénomènes complexes. À partir d'un mot initial et d'opérations de remplacement, on arrive à des mots compliqués. Lorsque l'on « dessine » ces mots, on obtient de superbes figures fractales. Le « L » vient du botaniste A. Lindenmayer qui a inventé les L-systèmes afin de modéliser les plantes.



## Cours 1 (L-système).

Un **L-système** est la donnée d'un mot initial et de règles de remplacement. Voici un exemple avec le mot de départ et une seule règle :

**BgAdB**       $A \rightarrow ABA$

Le ***k*-ème itéré** du L-système s'obtient en appliquant *k* fois la substitution au mot de départ. Avec notre exemple :

- Première itération. Le mot de départ est **BgAdB**, la règle est  $A \rightarrow ABA$  : on remplace le **A** par **ABA**. On obtient le mot **BgABAdB**.
- Deuxième itération. On part du mot obtenu **BgABAdB**, on remplace les deux **A** par **ABA** : on obtient le mot **BgABABABAdB**.
- Le troisième itéré est **BgABABABABABABABAdB**, etc.

Lorsqu'il y a deux règles (ou plus) il faut les appliquer en même temps. Voici un exemple de

L-système à deux règles :

$$A \quad A \rightarrow BgA \quad B \rightarrow BB$$

Avec notre exemple :

- Première itération. Le mot de départ est **A**, on applique la première règle  $A \rightarrow BgA$  (la seconde règle ne s'applique pas, car il n'y a pas encore de **B**) : on obtient le mot **BgA**.
- Deuxième itération. On part du mot obtenu **BgA**, on remplace les **A** par **BgA** et en même temps les **B** par **BB** : on obtient le mot **BBBgBgA**.
- Le troisième itéré est **BBBBBgBBBgBgA**, etc.

## Cours 2 (Argument optionnel d'une fonction).

Je veux programmer une fonction qui trace un trait d'une longueur donnée, avec la possibilité de changer l'épaisseur du trait et la couleur.

Une méthode serait de définir une fonction par :

```
def tracer(longueur, epaisseur, couleur):
```

Je l'appellerais alors par exemple par :

```
tracer(100, 5, "blue"):
```

Mais comme mes traits auront le plupart du temps l'épaisseur 5 et la couleur bleue, je perds du temps et de la lisibilité en redonnant ces informations à chaque fois.

Avec Python il est possible de donner des arguments optionnels. Voici une meilleure façon de faire en donnant des valeurs par défaut :

```
def tracer(longueur, epaisseur=5, couleur="blue"):
```

- La commande `tracer(100)` trace mon trait, et comme je n'ai précisé que la longueur, les arguments `epaisseur` et `couleur` prennent les valeurs par défaut (5 et bleu).
- La commande `tracer(100, epaisseur=10)` trace mon trait avec une nouvelle épaisseur (la couleur est celle par défaut).
- La commande `tracer(100, couleur="red")` trace mon trait avec une nouvelle couleur (l'épaisseur est celle par défaut).
- La commande `tracer(100, epaisseur=10, couleur="red")` trace mon trait avec une nouvelle épaisseur et une nouvelle couleur.
- Voici aussi ce que tu peux utiliser :
  - `tracer(100, 10, "red")` : ne pas préciser les noms des options si on fait attention à l'ordre.
  - `tracer(couleur="red", epaisseur=10, longueur=100)` : on peut nommer n'importe quelle variable, les variables nommées peuvent passer en argument dans n'importe quel ordre !

## Activité 1 (Tracer un mot).

*Objectifs : tracer un dessin à partir d'un « mot ». Chaque caractère correspond à une instruction de la tortue.*

On te donne un mot (par exemple **AgAdAAAdAdA**) dans lequel chaque lettre (lues de gauche à droite) correspond à une instruction pour la tortue Python.

- **A** ou **B** : avance d'une quantité fixée (en traçant),



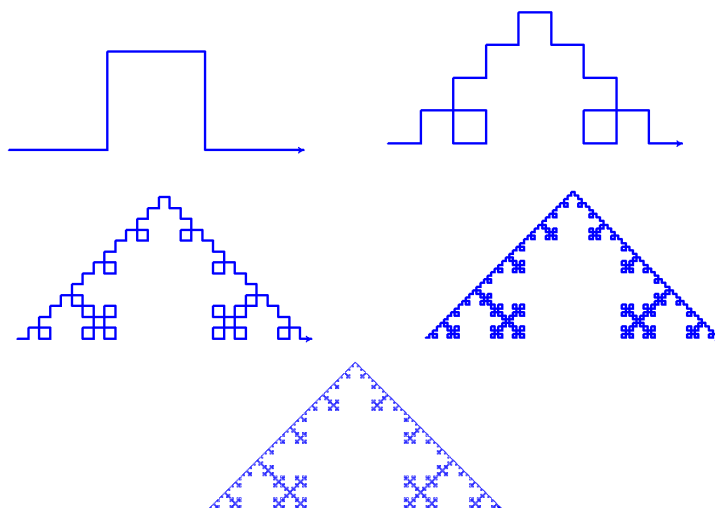
- pour  $k = 2$ , la fonction renvoie :  

$$\text{AgAdAdAgAgAgAdAdAgAdAdAgAdAdAgAdAdAgAgAgAdAdAgA}$$
- pour  $k = 3$ , la fonction renvoie :  $\text{AgAdAdAgAgA} \dots$  un mot de 249 lettres.

3. Trace les premières images du flocon de Koch donné comme ci-dessus par :

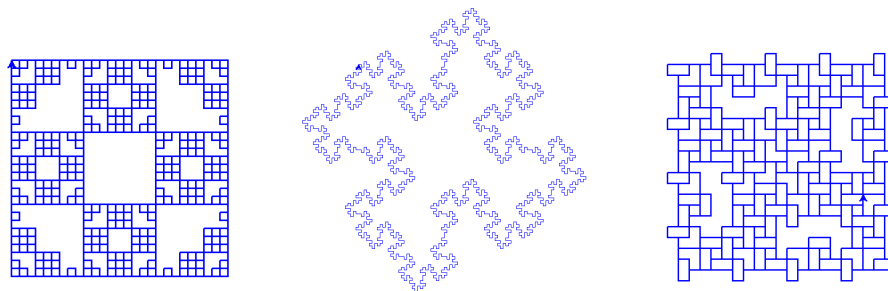
départ : **A**      règle :  $\text{A} \rightarrow \text{AgAdAdAgA}$

Ici les images pour  $k = 1$  jusqu'à  $k = 5$ . Pour  $k = 1$ , le mot est **AgAdAdAgA** et tu peux vérifier le tracé sur la première image.



4. Trace d'autres figures fractales à partir des L-systèmes suivants. Pour tous ces exemples le mot de départ est "AdAdAdA" (un carré) et la règle est à choisir parmi :

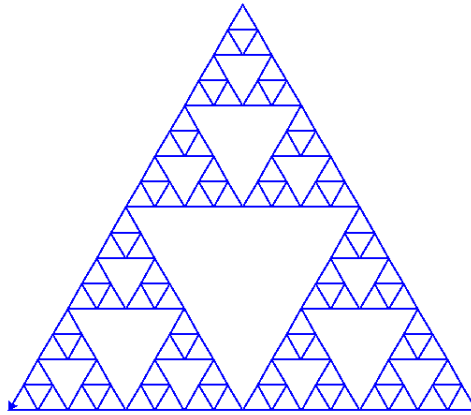
- ("A", " AdAgAgAAAdAdAgA")
- ("A", "AgAAAdAAAdAdAgAgAAAdAdAgAgAAAdAdAgA")
- ("A", "AAAdAdAdAdAA")
- ("A", "AAAdAddAdA")
- ("A", "AAAdAdAdAdAdAgA")
- ("A", "AAAdAgAdAdAA")
- ("A", "AdAAAddAdA")
- ("A", "AdAgAdAdAdA")



*Invente et trace tes propres L-systèmes !*

**Activité 3** (Deux règles – Triangle de Sierpinski).

*Objectifs : calculer des L-systèmes plus compliqués en autorisant cette fois deux règles de remplacement au lieu d'une seule.*



1. Programme une fonction `remplacer_2(mot, lettre1, motif1, lettre2, motif2)` qui remplace une première lettre par un motif et une seconde lettre par un autre.

Par exemple avec `mot = "AdBgA"`, `remplacer_2(mot, "A", "ABg", "B", "Bd")` renvoie le mot `ABgdBdgABg` : chaque lettre **A** a été remplacée par le motif **ABg** et en même temps chaque lettre **B** a été remplacée par le **Bd**.

*Attention !* Il ne faut pas obtenir `ABgdgBdgABdg`. Si c'est le cas c'est que tu as utilisé la fonction `remplacer_1()` pour d'abord remplacer les A, puis une seconde fois pour les B (mais après le premier remplacement de nouveaux B sont apparus). Il faut reprogrammer une nouvelle fonction pour éviter cela.

2. Programme une fonction `iterer_1systeme_2(depart, regle1, regle2, k)` qui calcule le  $k$ -ème itéré du L-système associé au mot initial `depart` selon les règles `regle1` et `regle2`. Par exemple, avec :

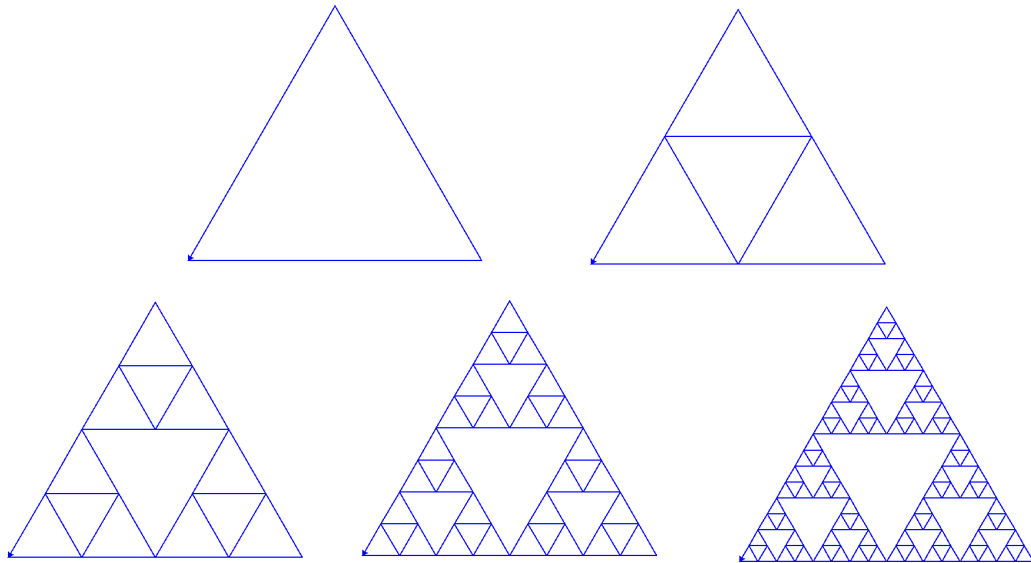
- `depart = "AdBdB"`
- `regle1 = ("A", "AdBgAgBdA")` c'est-à-dire **A** → **AdBgAgBdA**
- `regle2 = ("B", "BB")` c'est-à-dire **B** → **BB**
- pour  $k = 0$ , la fonction renvoie le mot de départ `AdBdB`,
- pour  $k = 1$ , la fonction renvoie `AdBgAgBdAdBBdBB`,
- pour  $k = 2$ , la fonction renvoie :

`AdBgAgBdAdBBgAdBgAgBdAgBBdAdBgAgBdAdBBBBdBBBB`

3. Trace les premières images du triangle de Sierpinski donné comme ci-dessus par :

départ : **AdBdB**      règles : **A** → **AdBgAgBdA**    **B** → **BB**

L'angle est de  $-120$  degrés. Voici les images pour  $k = 0$  jusqu'à  $k = 4$ .



4. Trace d'autres figures fractales à partir des L-systèmes suivants.

- La courbe du dragon :

depart="AX" regle1=("X","XgYAg") regle2=("Y","dAXdY")

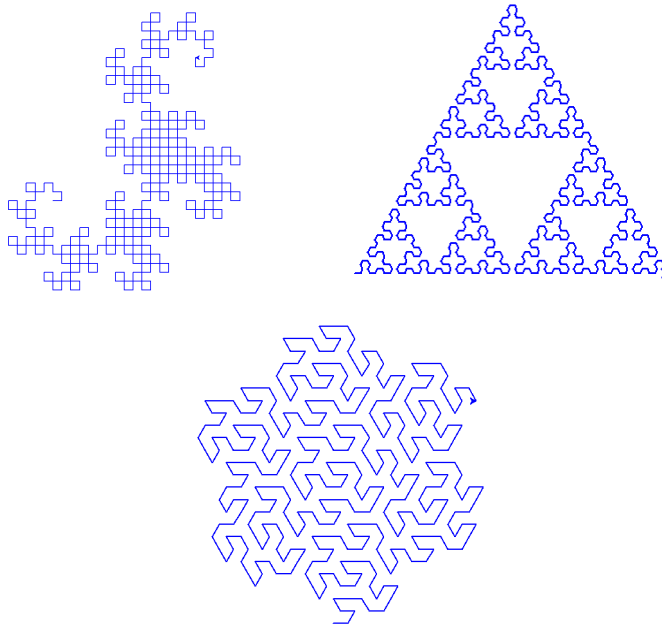
Les lettres X et Y ne correspondent à aucune action.

- Une variante du triangle de Sierpinski, avec angle = 60 :

depart="A" regle1=("A","BdAdB") regle2=("B","AgBgA")

- La courbe de Gosper, avec angle = 60 :

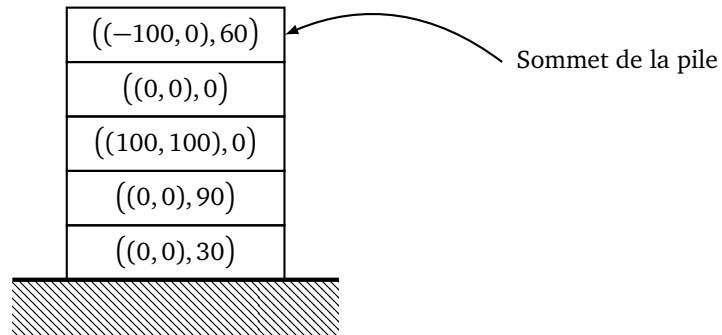
depart="A" regle1=("A","AgBggBdAddAAdBg") regle2=("B","dAgBBggBgAddAdB")



*Invente et trace tes propres L-systèmes avec deux règles !*

**Cours 3 (Piles).**

Une **pile** est une zone de stockage temporaire. Les détails sont dans la fiche « Calculatrice polonaise – Piles ». Voici juste quelques rappels.

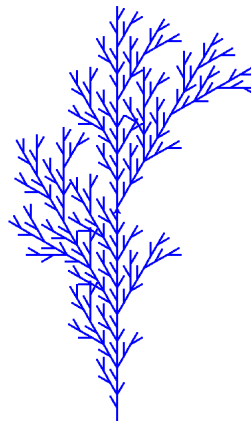


**Une pile**

- Une pile c'est comme un pile d'assiettes ; on pose des éléments un par un au-dessus de la pile ; on retire les éléments un par un, également à partir du dessus. C'est le principe : « dernier arrivé, premier parti » (*last in, first out*).
- On modélise une pile par une liste.
- Au départ la pile est vide : `pile = []`.
- **Empiler.** On ajoute les éléments en fin de liste : `pile = pile + [element]`.
- **Dépiler.** On retire un élément par la commande `pop()` :  
`element = pile.pop()`  
 qui renvoie le dernier élément de la pile et le supprime de la liste.
- Sur le dessin et dans l'activité suivante, les éléments de la pile sont du type  $((x, y), \theta)$  qui stockeront un état de la tortue :  $(x, y)$  est la position et  $\theta$  sa direction.

**Activité 4 (L-système et tortue à pile).**

*Objectifs : améliorer nos tracés en autorisant d'avancer sans tracer et aussi à l'aide d'une sorte de retour en arrière, afin de tracer des plantes.*

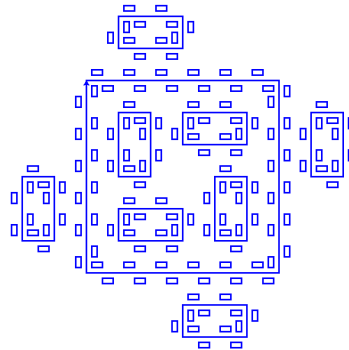


### 1. Avancer sans tracer.

Augmente les possibilités en autorisant la tortue à avancer sans tracer de trait, lorsque l'instruction est la lettre est **a** (en minuscule). (Il suffit de modifier la fonction `trace_lsysteme()`.)

Trace alors le L-système suivant :

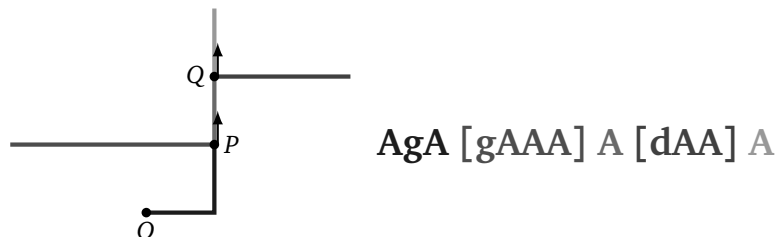
- `depart = "AdAdAdA"`
- `regle1 = ("A", "AgadAAgAgAAgAagAAadagAAAdAdAAAdAdAAA")`
- `regle2 = ("a", "aaaaaa")`



### 2. Retour en arrière.

On autorise maintenant des crochets dans nos mots. Par exemple **AgA[gAAA]A[dAA]A**. Lorsque l'on rencontre un crochet ouvrant « [ », on mémorise la position de la tortue, puis les commandes entre crochets sont exécutées comme d'habitude, lorsque l'on trouve un crochet fermant « ] » on repart de la position mémorisée auparavant.

Comprenons l'exemple du tracé du **AgA [gAAA] A [dAA] A**.



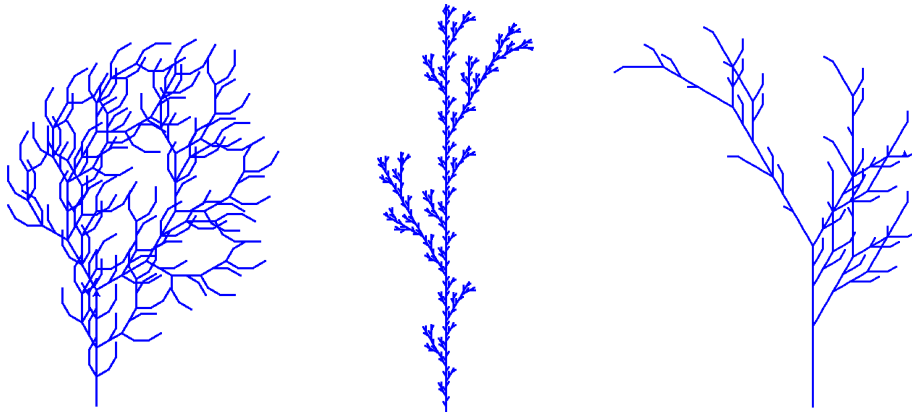
- **AgA** : on part du point *O*, on avance, on tourne, on avance.
- **[gAAA]** : on retient la position actuelle (le point *P*) et aussi la direction ; on tourne, on avance trois fois (on trace le segment rouge) ; à la fin on replace la tortue à la position *P* (sans tracer et avec la même direction que celle auparavant).
- **A** : depuis *P* on avance (segment vert).
- **[dAA]** : on retient la position *Q* et la direction, on tourne et on trace le segment violet. On revient en *Q* avec l'ancienne direction.
- **A** : depuis *Q* on trace le dernier segment.

Voici comment tracer un mot contenant des crochets à l'aide d'une pile :

- Au départ la pile est vide.
- On lit un par un les caractères du mot. Les actions sont les mêmes qu'auparavant.



- Si le caractère est le crochet ouvrant « [ » alors on ajoute à la pile la position et la direction courante de la tortue  $((x, y), \theta)$  que l'on obtient par `position()`, `heading()`.
  - Si le caractère est le crochet fermant « ] » alors dépiler (c'est-à-dire lire l'élément du haut de la pile et le retirer). Mettre la position de la tortue et l'angle avec les valeurs lues, utiliser `goto()` et `setheading()`.
3. Trace les L-systèmes suivants, où l'on donne le mot de départ et la règle (ou les règles). L'angle est à choisir entre 20 et 30 degrés.
- "A" ("A", "A[gA]A[dA][A]")
  - "A" ("A", "A[gA]A[dA]A")
  - "A" ("A", "AAd[dAgAgA]g[gAdAdA]")
  - "X" ("X", "A[gX][X]A[gX]dAX") ("A", "AA")
  - "X" ("X", "A[gX]A[dX]AX") ("A", "AA")
  - "X" ("X", "Ad[[X]gX]gA[gAX]dX") ("A", "AA")



*Invente ta propre plante !*