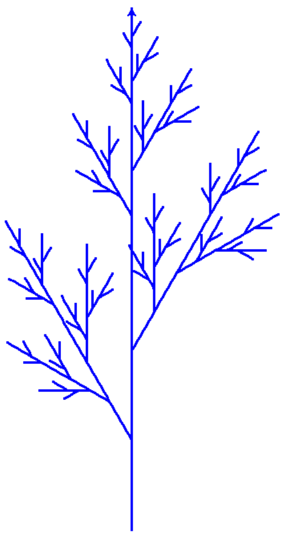
L-systems

The L-systems offer a very simple way to code complex phenomena. From an initial word and replacement operations, we arrive at complicated words. When you "draw" these words, you get beautiful fractal figures. The "L" comes from the botanist A. Lindenmayer who invented the L-systems to model plants.



Lesson 1 (L-system)**.**
A ***L-system*** is the data of an initial word and replacement rules. Here is an example with the starting word and only one rule:

$$\textbf{BlArB} \qquad \textbf{A} \to \textbf{ABA}$$

The ***k-iteration*** of the L-system is obtained by applying k times the substitution to the starting word. With our example:

- First iteration. The starting word is **BlArB**, the rule is **A** → **ABA**: we replace the **A** by **ABA**. We get the word **BlABArB**.
- Second iteration. We start from the word obtained **BlABArB**, we replace the two **A** by **ABA**: we get the word **BlABABABArB**.
- The third iteration is **BlABABABABABABABArB**, etc.

When there are two (or more) rules, they must be applied at the same time. Here is an example of a two-rules L-system:

$$\textbf{A} \qquad \textbf{A} \to \textbf{BlA} \qquad \textbf{B} \to \textbf{BB}$$

With our example:

- First iteration. The starting word is **A**, we apply the first rule **A** → **BlA** (the second rule does not apply, because there is no **B** yet): we get the word **BlA**.

- Second iteration. We start from the word obtained **BlA**, we replace the **A** by **BlA** and at the same time the **B** by **BB**: we get the word **BBlBlA**.
- The third iteration is **BBBBlBBlBlA**, etc.

Lesson 2 (Optional argument for a function).
I want to program a function that draws a line of a given length, with the possibility to change the thickness of the line and the color.
One method would be to define a function by:

```
def draw(length, width, color):
```

I would then call it for example by:

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

But since my features will most of the time have a thickness of 5 and a blue color, I lose time and legibility by giving this information back each time.

With `Python` it is possible to give optional arguments. Here is a better way to do this by giving default values:

```
def draw(length, width=5, color="blue"):
```

- The command `draw(100)` draws my line, and as I only specified the length, the arguments `width` and `color` take the default values (5 and blue).
- The command `draw(100, width=10)` draws my line with a new thickness (the color is the default one).
- The command `draw(100, color="red")` draws my line with a new color (the thickness is the default).
- The command `draw(100, width=10, color="red")` draws my line with a new thickness and a new color.
- Here is also what you can use:
  - `draw(100, 10, "red")`: do not specify the names of the options if you pay attention to the order.
  - `draw(color="red", width=10, length=100)`: you can name any variable, the named variables can pass as arguments in any order.

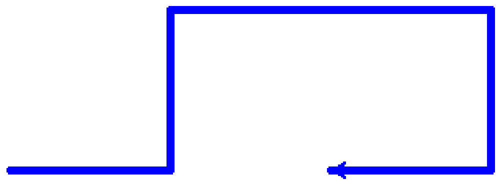
Activity 1 (Draw a word).

Goal: make a drawing from a "word". Each character corresponds to a turtle instruction.

You are given a word (for example **AlArAArArA**) in which each letter (read from left to right) corresponds to an instruction for the `Python` turtle.

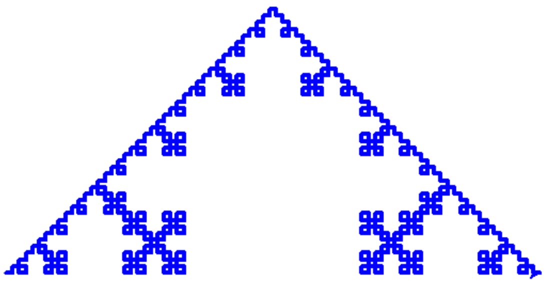
- **A** or **B**: advance of a fixed quantity (by tracing),
- **l**: turn left, without moving forward, by a fixed angle (most often 90 degrees),
- **r**: turns right by a fixed angle.

The other characters do not do anything. (More commands will be added later on.)
Program a function `draw_lsystem(word,angle=90,scale=1)` which displays the drawing corresponding to the letters of `word`. By default the angle is 90 degrees, and each time you advance, it is of 100×`scale`.
For example: `draw_lsystem("AlArAArArA")` displays this:



Activity 2 (Only one rule – Koch's snowflake)**.**

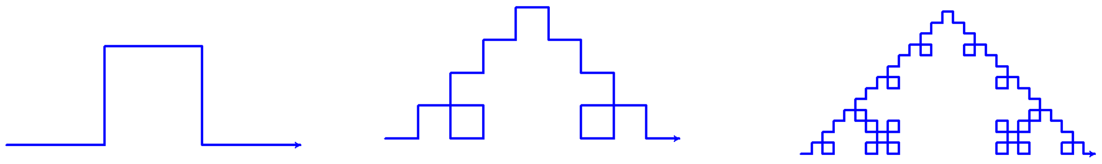
Goal: draw the Koch snowflake from a word obtained by iterations.

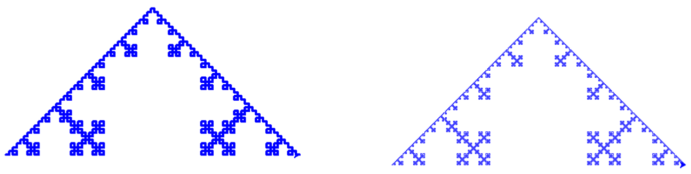


1. Program a function `replace_1(word,letter,pattern)` that replaces a letter with a pattern in a word.
   For example with `word = "ArAAl"`, `replace_1(word,"A","Al")` returns the word `AlrAlAll`: each letter **A** has been replaced by the pattern **Al**.
2. Program a function `iterate_lsystem_1(start,rule,k)` which calculates the k-iteration of the L-system associated with the initial word `start` according to the rule `rule` which contains the couple formed by the letter and its replacement pattern. For example, with:
   - `start = "A"`
   - `rule = ("A","AlArArAlA")` i.e. **A** → **AlArArAlA**
   - for `k = 0`, the function returns the starting word `A`,
   - for `k = 1`, the function returns `AlArArAlA`,
   - for `k = 2`, the function returns :
     `AlArArAlAlAlArArAlArAlArArAlArAlArArAlAlAlArArAlA`
   - for `k = 3`, the function returns: `AlArArAlAlAl...` a word of 249 letters.
3. Trace the first images of the Koch's snowflake given as above by:

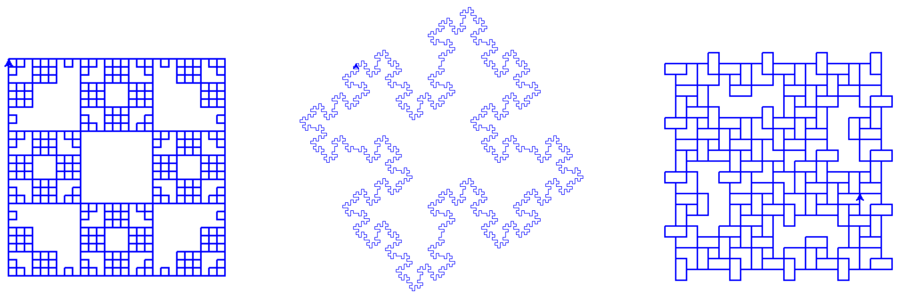
   start: **A** rule: **A** → **AlArArAlA**

   Here the images for $k = 1$ up to $k = 5$. For $k = 1$, the word is **AlArArAlA** and you can check the trace on the first image.





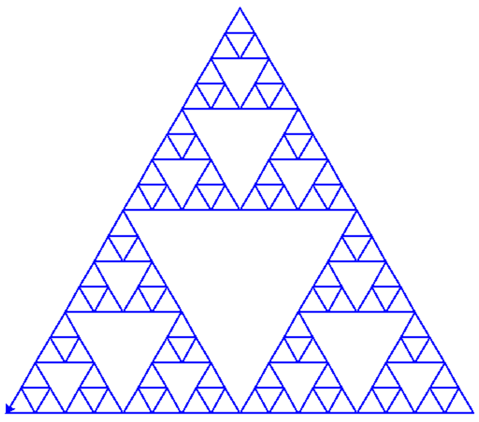
4. Trace other fractal figures from the following L-systems. For all these examples the starting word is `"ArArArA"` (a square) and the rule is to be chosen among:
   - `("A","ArAlAlAArArAlA")`
   - `("A","AlAArAArArAlAlAArArAlAlAAlAArA")`
   - `("A","AArArArArAA")`
   - `("A","AArArrArA")`
   - `("A","AArArArArArAlA")`
   - `("A","AArAlArArAA")`
   - `("A","ArAArrArA")`
   - `("A","ArAlArArA")`



Invent and trace your own L-systems!

Activity 3 (Two rules – Sierpinski triangle).

Goal: calculate more complicated L-systems by allowing two replacement rules instead of one.



1. Program a function `replace_2(word,letter1,pattern1,letter2,pattern2)` that replaces a first letter with a pattern and a second letter with another.

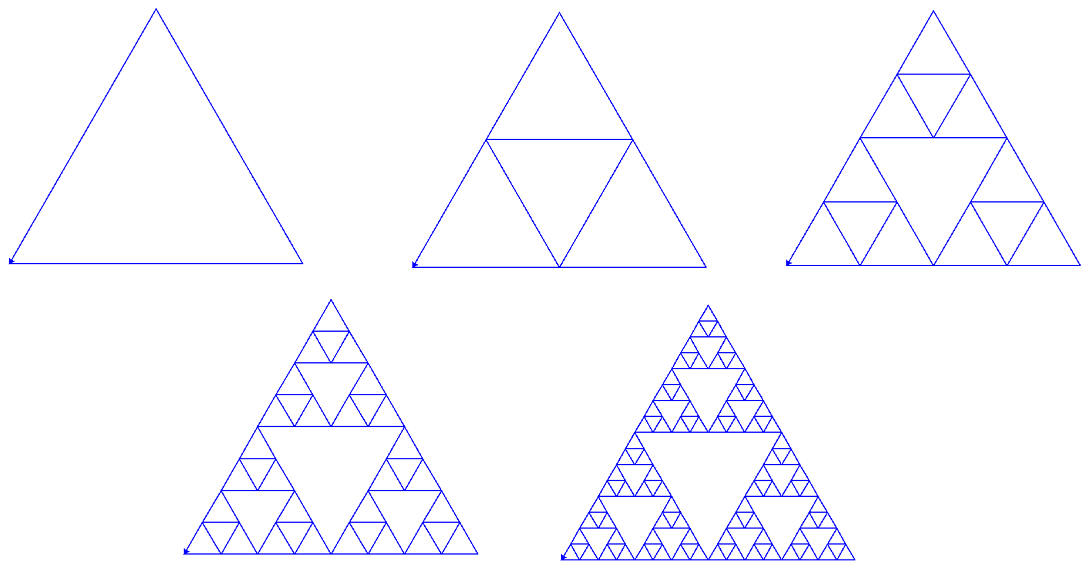
For example with `word = "ArBlA"`, `replace_2(word,"A","ABl","B","Br")` returns the word `ABlrBrlABl`: each letter **A** has been replaced by the pattern **ABl** and at the same time each letter **B** has been replaced by the **Br**.
Warning! Do not get `ABrlrBrlABrl`. If this is the case, it is because you used the `replace_1()` function to first replace the `A`, then a second time for the `B` (but after the first replacement new `B` appeared). A new function must be reprogrammed to avoid this.

2. Program a function `iterate_lsystems_2(start,rule1,rule2,k)` which calculates the k-iteration of the L-system associated with the initial word `start` according to the rules `rule1` and `rule2`. For example, with:
   - `start = "ArBrB"`
   - `rule1 = ("A","ArBlAlBrA")` i.e. **A** $\longrightarrow$ **ArBlAlBrA**
   - `rule2 = ("B","BB")` i.e. **B** $\longrightarrow$ **BB**
   - for `k = 0`, the function returns the starting word `ArBrB`,
   - for `k = 1`, the function returns `ArBlAlBrArBBrBB`,
   - for `k = 2`, the function returns:

     `ArBlAlBrArBBlArBlAlBrAlBBrArBlAlBrArBBBBrBBBB`
3. Trace the first pictures of the Sierpinski triangle given as above by:

   start: **ArBrB** rules: **A** $\longrightarrow$ **ArBlAlBrA** **B** $\longrightarrow$ **BB**

   The angle is -120 degrees. Here are the images for $k=0$ up to $k=4$.



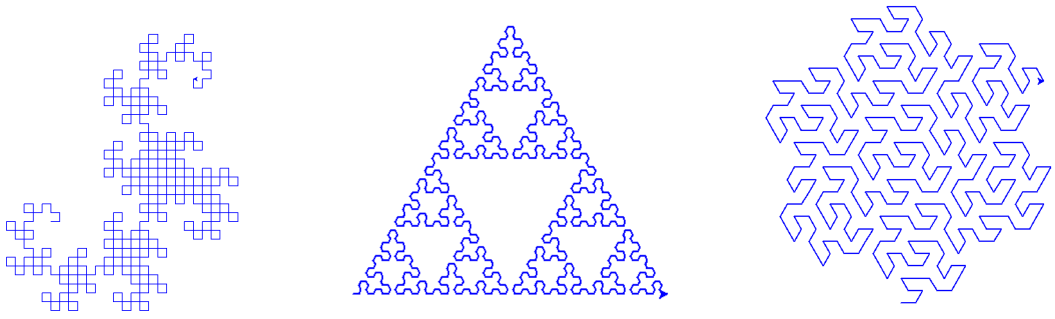
4. Trace other fractal figures from the following L-systems.
   - The dragon curve:

     `start="AX" rule1=("X","XlYAl") rule2=("Y","rAXrY")`

     The letters `X` and `Y` do not correspond to any action.
   - A variant of the Sierpinski triangle, with `angle = 60` :

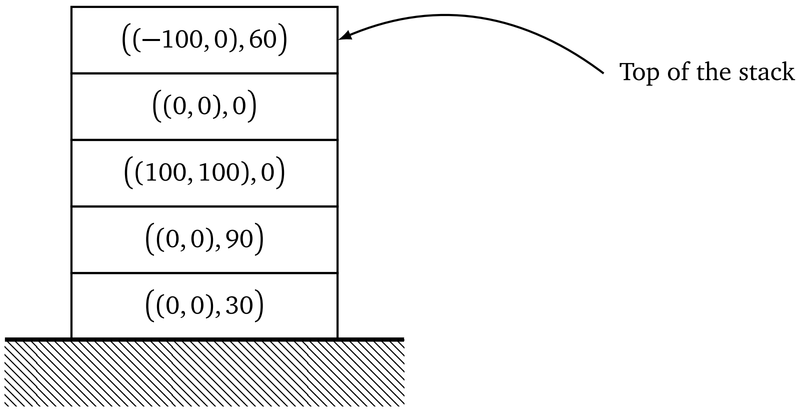
     `start="A" rule1=("A","BrArB") rule2=("B","AlBlA")`
   - The Gosper curve, with `angle = 60`:

     ```
     start="A"
     rule1=("A","AlBllBrArrAArBl")
     rule2=("B","rAlBBllBlArrArB")
     ```



Invent and trace your own L-systems with two rules!

Lesson 3 (Stacks).
A *stack* is a temporary storage area. Details are in the chapter “Polish calculator – Stacks”. Here are just a few reminders.

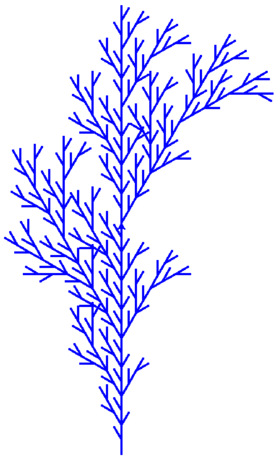


A stack

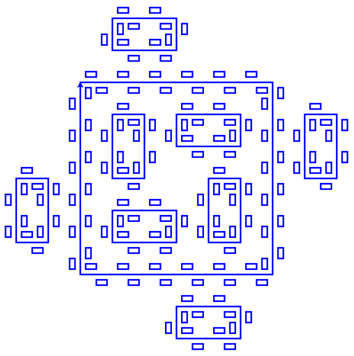
- A stack is like a stack of plates; elements are placed one by one above the stack; the elements are removed one by one, always from the top. It’s the principle: “last in, first out”.
- We model a stack by a list.
- At the beginning the stack is empty: `stack = []`.
- **Push.** We add the items at the end of the list: `stack.append(element)` or `stack = stack + [element]`.
- **Pop.** An item is removed by the command `pop()`:
  ```
  element = stack.pop()
  ```
  which returns the last item in the stack and removes it from the list.
- On the drawing and in the next activity, the elements of the stack are of the type $\big((x,y),\theta\big)$ that will store a state of the turtle: (x,y) is the position and θ its direction.

Activity 4 (L-system, stack and turtle).

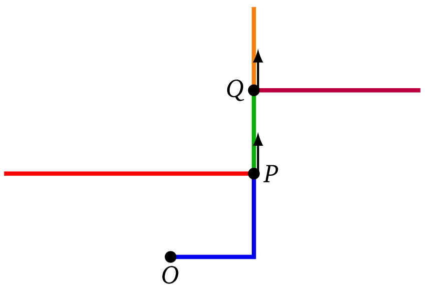
Goal: improve our tracings by allowing us to move forward without tracing and also by using a kind of flashback, to trace plants.



1. **Forward without tracing.**
   Increase the possibilities by allowing the turtle to move forward without drawing a line, when the instruction is the letter **a** (in lowercase). (It is sufficient to modify the function `trace_lsystems()`.)
   Then trace the following L-system:
   - `start = "ArArArA"`
   - `rule1 = ("A","AlarAAlAlAAlAalAAralAArArAArAarAAA")`
   - `rule2 = ("a","aaaaaa")`



2. **Return back.**
   We now allow square brackets in our words. For example **AlA[lAAA]A[rAA]A**. When you encounter a opening bracket “[”, you memorize the position of the turtle, then the commands in brackets are executed as usual, when you find the closing bracket “]” you go back to the position stored before.
   Let us understand the example of the route of the **AlA [lAAA] A [rAA] A**



AlA [lAAA] A [rAA] A

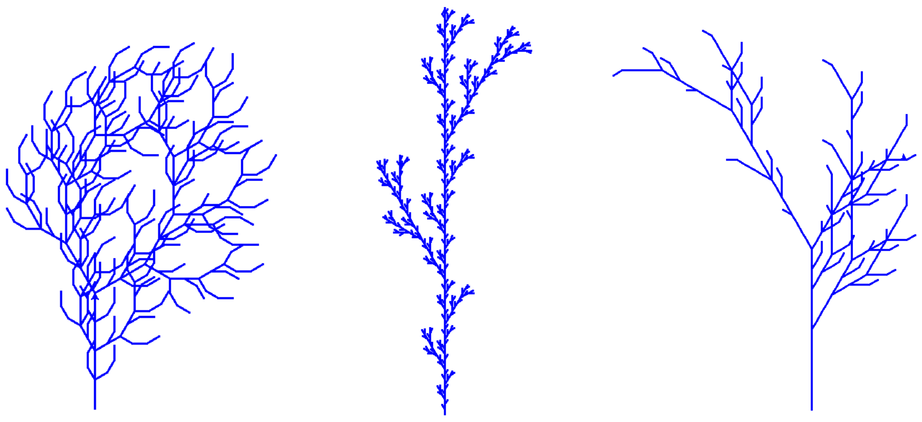
- **AlA**: we start from the point O, we move forward, we turn, we move forward.

- [lAAA]: we retain the current position (the point P) and also the direction; we turn, we advance three times (we trace the red segment); at the end we return the turtle to the position P (without tracing and with the same direction as before).
- A: from P we advance (green segment).
- [rAA]: we retain the position Q and the direction, we turn and we trace the purple segment. We come back in Q with the old state.
- A: from Q we trace the last segment.

Here is how to draw a word containing brackets using a stack:

- At the beginning the stack is empty.
- We read one by one the characters of the word. The actions are the same as before.
- If the character is the opening bracket “[” then add to the stack the current position and direction of the turtle $\big((x,y),\theta\big)$ that you get by `(position(), heading())`.
- If the character is the closing bracket “]” then pop (i.e. read the top element of the stack and remove it). Put the position of the turtle and the angle with the read values, use `goto()` and `setheading()`.

3. Trace the following L-systems, where the starting word and rule (or rules) are given. The angle is to be chosen between 20 and 30 degrees.
    - `"A"   ("A","A[lA]A[rA][A]")`
    - `"A"   ("A","A[lA]A[rA]A")`
    - `"A"   ("A","AAr[rAlAlA]l[lArArA]")`
    - `"X"   ("X","A[lX][X]A[lX]rAX")   ("A","AA")`
    - `"X"   ("X","A[lX]A[rX]AX")   ("A","AA")`
    - `"X"   ("X","Ar[[X]lX]lA[lAX]rX")   ("A","AA")`



Invent your own plant!