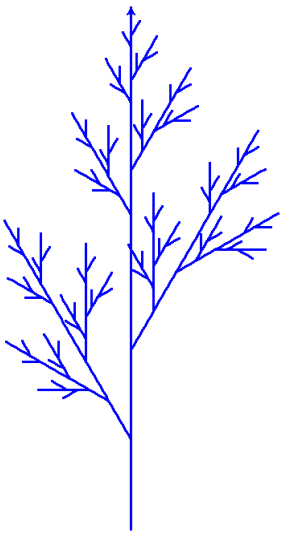
L-systems

L-systems offer a very simple way to code complex phenomena. From an initial word and a number of replacement operation, 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 L-systems to model plants.



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

BlArB $\qquad$ **A** $\to$ **ABA**

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

- First iteration. The starting word is **BlArB**, the rule is **A** $\to$ **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-rule L-system:

A $\qquad$ **A** $\to$ **BlA** $\qquad$ **B** $\to$ **BB**

With our example:

- First iteration. The starting word is **A**, we apply the first rule **A** $\to$ **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 like this:

```
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 each time.

With `Python` it is possible to create optional arguments. There is a way to use optional arguments by giving the function 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` get 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 one).
- The command `draw(100, width=10, color="red")` draws my line with a new thickness and a new color.
- We can also use:
  - `draw(100, 10, "red")`: no need to specify the names of the arguments if you maintain the order.
  - `draw(color="red", width=10, length=100)`: if you name the arguments, then you can pass them in any order.

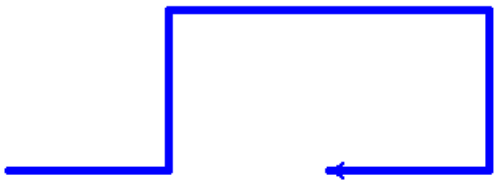
Activity 1 (Draw a word)**.**

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 `Python`'s turtle.

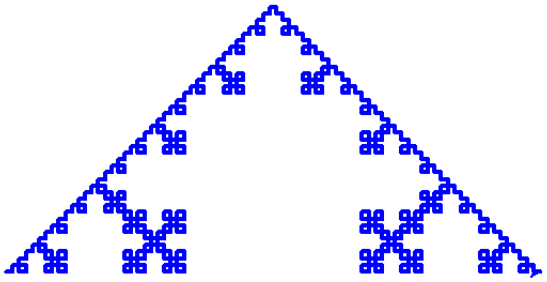
- **A** or **B**: advance by a fixed distance (by tracing),
- **l**: turn left, without moving forward, by a fixed angle (most often 90 degrees),
- **r**: turns right by a fixed angle.

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

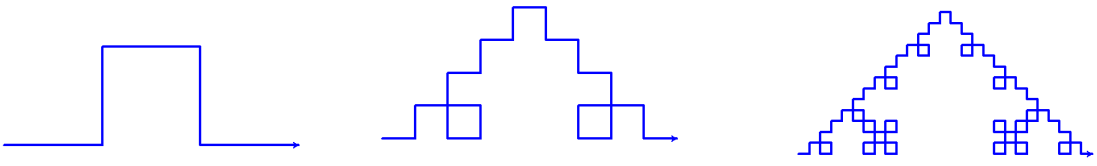


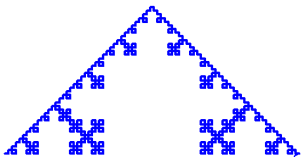
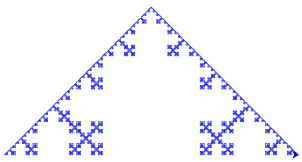
Activity 2 (Only one rule – Koch's snowflake)**.**

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

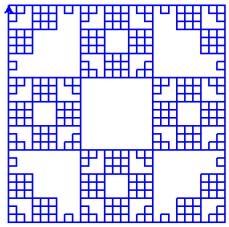
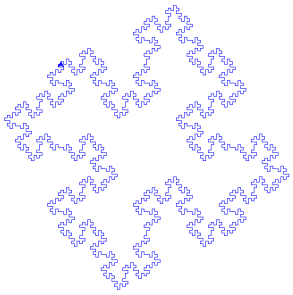
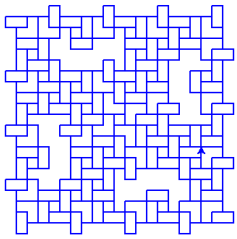


1. Program a `replace_1(word,letter,pattern)` function that replaces a letter with a pattern in a word.
   For example with `word = "ArAAl"`, `replace_1(word,"A","Al")` returns the word `AlrAlAll`: each letter **A** has been replaced by the pattern **Al**.
2. Program an `iterate_lsystem_1(start,rule,k)` function which calculates the k-iteration of the L-system associated with the initial word `start` according to the rule `rule` which contains the pair 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 are the images for $k = 1$ up to $k = 5$. For $k = 1$, the word is **AlArArAlA**, you can draw yourself and confirm the trace of the first image.



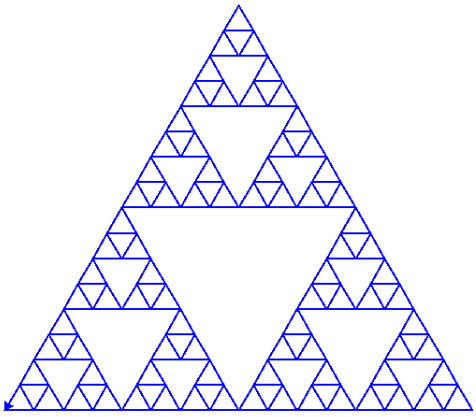
4. Trace other fractal figures from the following L-systems. For all these examples the starting word is `"ArArArA"` (a square) and the rule is to be chosen among:
   - `("A","ArAlAlAArArAlA")`
   - `("A","AlAArAArArAlAlAArArAlAlAAlAArA")`
   - `("A","AArArArArAA")`
   - `("A","AArArrArA")`
   - `("A","AArArArArArAlA")`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

Invent and trace your own L-systems!

Activity 3 (Two rules – Sierpinski triangle)**.**

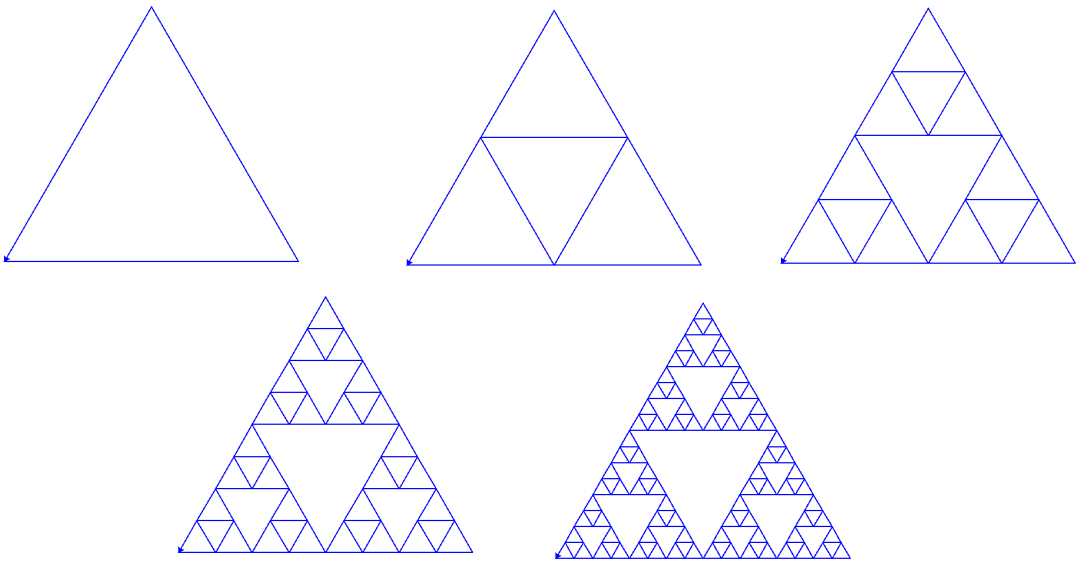
Goal: draw more complicated L-systems by allowing two replacement rules instead of one.



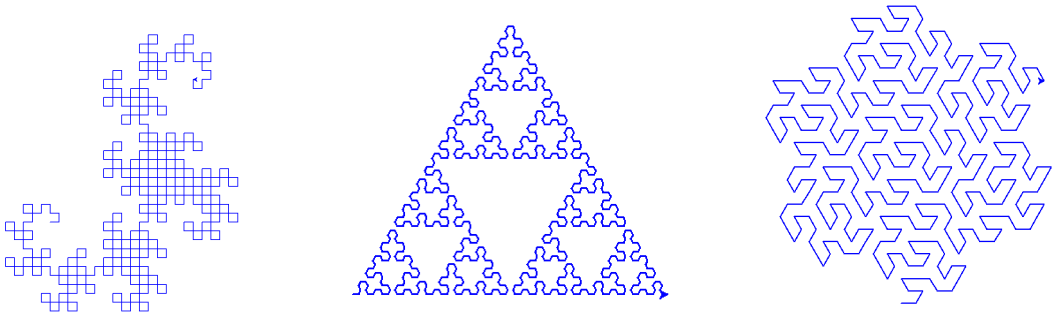
1. Program a `replace_2(word,letter1,pattern1,letter2,pattern2)` function that replaces the first letter with a pattern and the second letter with another pattern.

For example when `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 **Br**.
Warning! You should not get `ABrlrBrlABrl`. If this is the case, it is because you used the `replace_1()` function first to replace the `A`, then a second time to replace the `B` (but after the first replacement, new `B` appeared). A new function must be programmed to avoid this.

2. Program an `iterate_lsystems_2(start,rule1,rule2,k)` function 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** → **ArBlAlBrA**
   - `rule2 = ("B","BB")` i.e. **B** → **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** → **ArBlAlBrA** **B** → **BB**
   The angle is −120 degrees. Here are the images for $k = 0$ up to $k = 4$.

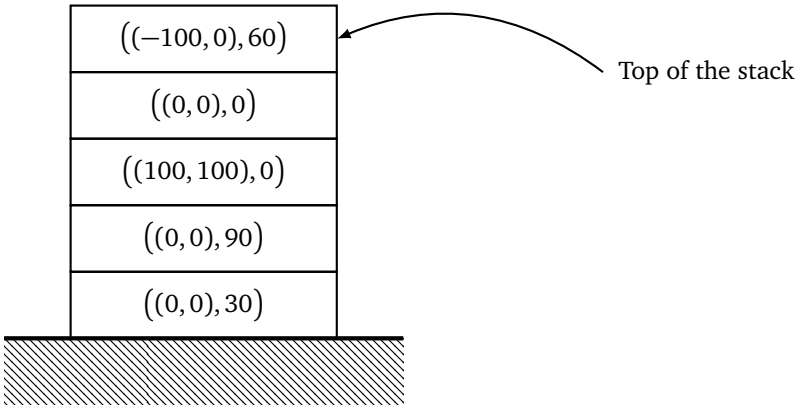


4. Trace other fractal figures from the following L-systems.
   - The dragon curve:
     `start="AX" rule1=("X","XlYAl") rule2=("Y","rAXrY")`
     The letters `X` and `Y` do not correspond to an action.
   - A variant of the Sierpinski triangle, where `angle = 60`:
     `start="A" rule1=("A","BrArB") rule2=("B","AlBlA")`
   - The Gosper curve, where `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 "Polish calculator – Stacks" chapter. Here are just a few reminders.

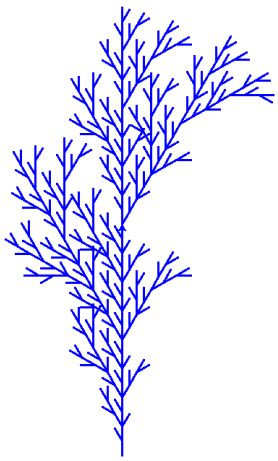


A stack

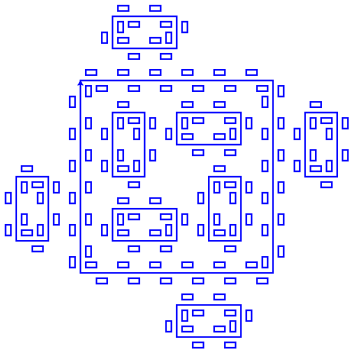
- A stack is like a stack of plates; elements are placed one by one to the top of the stack; the elements are removed one by one, always from the top. It follows the "last in, first out" principle.
- We model a stack using a list.
- At the beginning the stack is empty: `stack = []`.
- **Push.** We add the items to the end of the list: `stack.append(element)` or `stack = stack + [element]`.
- **Pop.** An item is removed by using the `pop()` command:
  ```
  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 type $\big((x, y), \theta\big)$ that will store a state of the turtle: (x, y) is the position and θ is its direction.

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

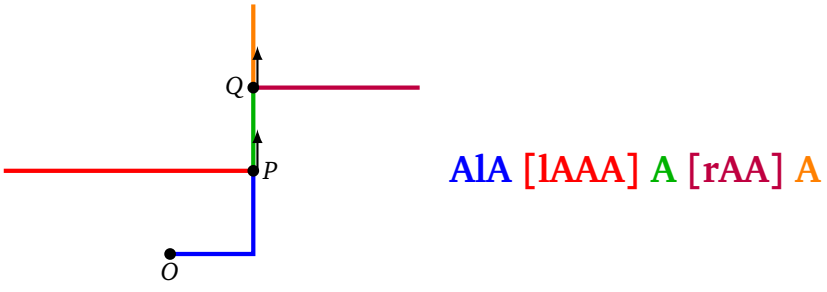
Goal: improve our drawings by allowing us to move forward without tracing and also by using a kind of flashback, to trace plants.



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 `trace_lsystems()` function.)
   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 store the position of the turtle, then the commands in brackets are executed as usual, when you reach the closing bracket "]" you go back to the stored position.
   Let us work through the example: **AlA [lAAA] A [rAA] A**



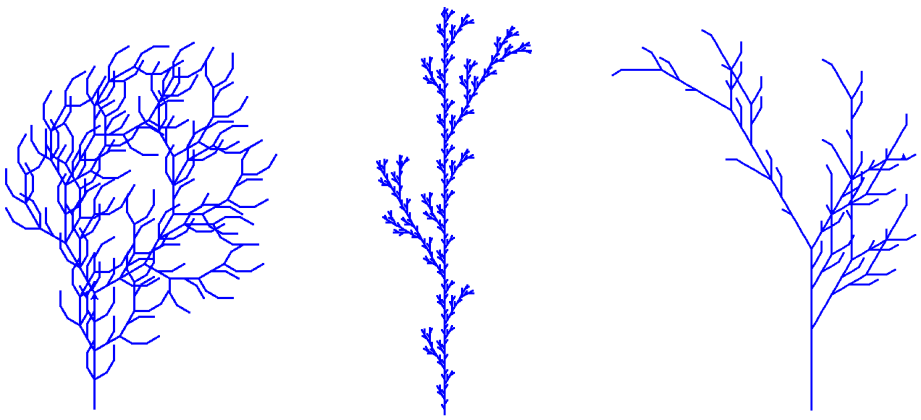
   - **AlA**: we start from the point O, we move forward, we turn, we move forward.

- **[lAAA]**: we store the current position (the point P) and 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 store the position Q and the direction, we turn and we trace the purple segment. We come back to 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 the characters of the word one by one. The actions are the same as before.
- If the character is the opening bracket "[" then add the current position and direction of the turtle $\big((x,y),\theta\big)$ to the stack. You get $\big((x,y),\theta\big)$ by `(position(), heading())`.
- If the character is the closing bracket "]" then pop (i.e. read the top element of the stack and remove it). Set the position of the turtle and the angle to 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!