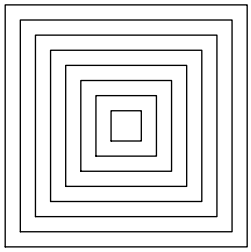
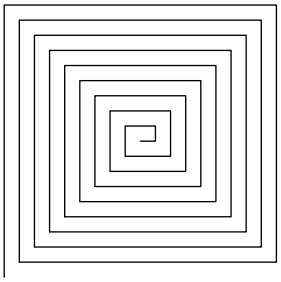
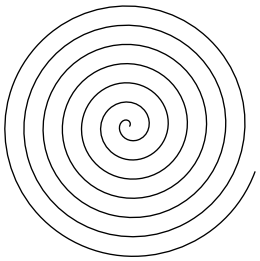
A. Нарисовать концентрические квадраты.



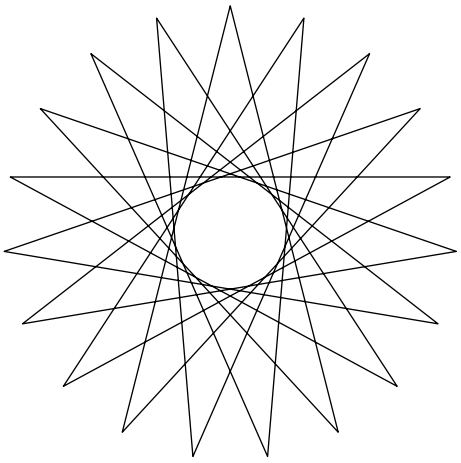
B. Нарисовать квадратную спираль.



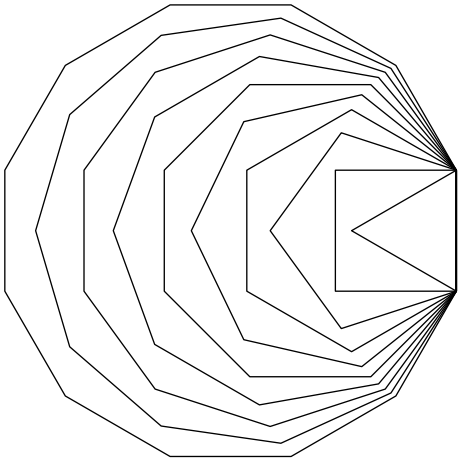
C. Нарисовать архимедову спираль.



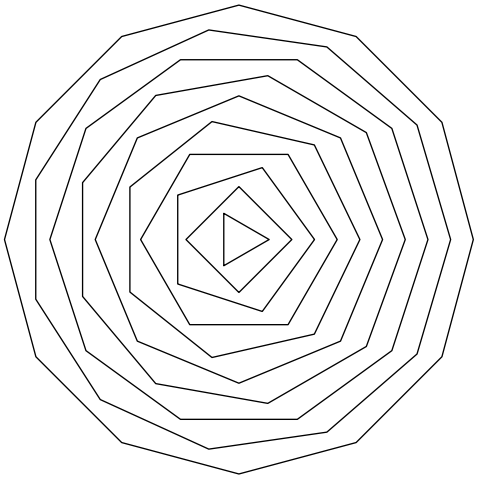
D. Нарисовать n-лучевую звезду.



E. Нарисовать несколько правильных многоугольников, построенных на одной и той же стороне.



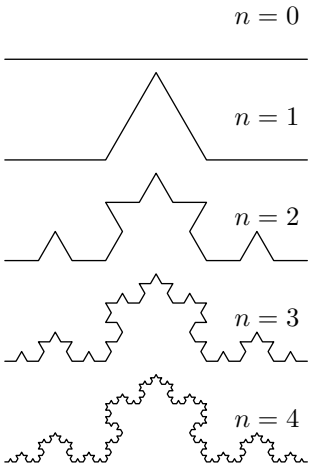
F. Нарисовать несколько правильных многоугольников, так, чтобы хотя бы одна вершина (не являющаяся общей ни для каких двух многоугольников) каждого лежала на одной прямой.



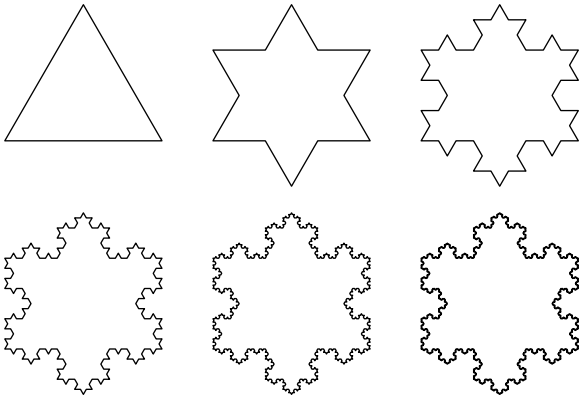
Использование рекурсии для рисования фракталов

При помощи рекурсивных функций можно рисовать более интересные объекты, например фракталы. Начнём с классического фрактала — кривой Коха.

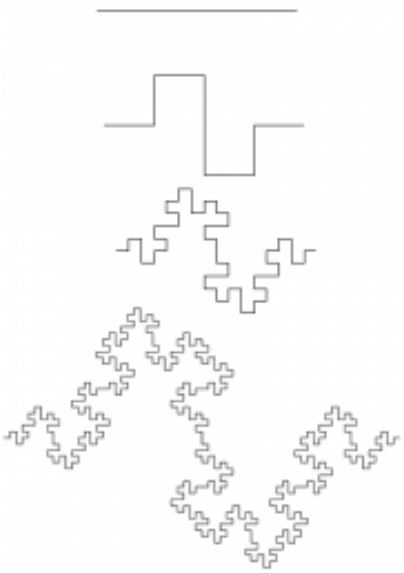
G. Кривая Коха нулевого порядка — это отрезок. Для построения кривой Коха порядка $n+1$ мы разбиваем каждый отрезок кривой Коха порядка n на три равные части и средний сегмент заменяем равносторонним треугольником без этого сегмента.



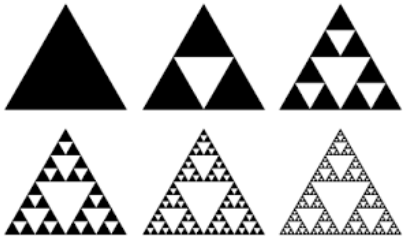
H. Похожим образом определяется т.н. снежинка Коха. Это равносторонний "треугольник", каждая сторона которого представляет собой кривую Коха одного и того же порядка.



I. Ещё один классический фрактал — кривая Минковского.

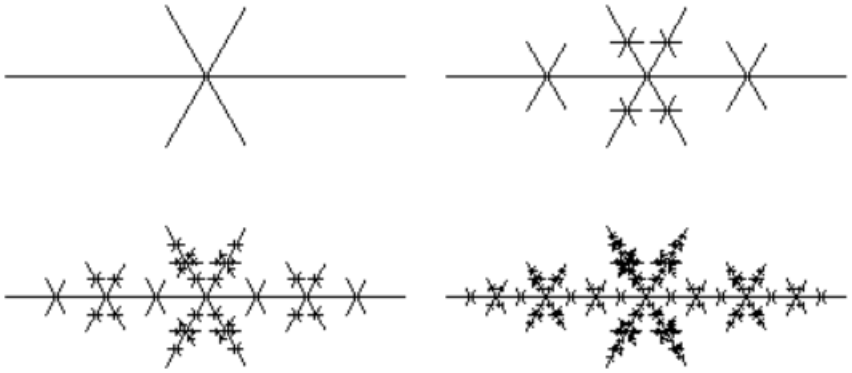


J. Треугольник Серпинского — последовательное “выкидывание” центрального треугольника. Двумерный аналог множества Кантора был предложен польским математиком Вацлавом Серпинским в 1915 году.



K. Ледяной фрактал.

Здесь можно поэкспериментировать с размерами “усов”, их формой (углом) и стороной, куда они растут.



Используя функцию, строящую такой отрезок (растущий в одну или обе стороны), напишите фукнцию, строящую правильный N-многоугольник, каждая сторона которого будет представлять собой такой отрезок.

В предыдущем разделе мы описывали рекурсивные функции, порождающие те или иные фрактальные (самоподобные) структуры. Можно пойти дальше и описать *язык*, порождающий последовательности команд управления черепашкой. Теперь надо написать только одну функцию-интерпретатор, переводящую предложения нашего языка на язык команд черепашки. А разные фракталы — это просто разные предложения на описанном нами языке.

Состояние черепашки описывается тройкой (x, y, α), где x, y - декартовы координаты черепашки, а α — угол (в градусах, отсчитываемый от положительного направления оси Ox), в направлении которого черепашка движется.

Для данных фиксированных значений d — величины шага и δ — величины изменения угла черепашка умеет выполнять команды, описываемые следующими символами:

`F` Сделать шаг длины d. Состояние черепашки станет равным (x', y'), где

$$x' = x + d \cdot cos\alpha, y' = y + d \cdot \sin\alpha$$

При этом между точками (x, y) и (x', y') рисуется отрезок.

`f` Сделать шаг длины d, без рисования отрезка.

`+` Повернуть налево на угол δ. Состояние черепашки меняется на $(x, y, \alpha + \delta)$. Поворот на положительный угол происходит против часовой стрелки.

`-` Повернуть направо на угол δ. Состояние черепашки меняется на $(x, y, \alpha - \delta)$.

Для порождения строк, управляющих черепашкой можно воспользоваться следующей конструкцией.

Пусть V — некоторый алфавит, V^+ — множество всех непустых слов над алфавитом V, V^* — множество всех слов над алфавитом V.

Определим L-систему как тройку $G = (V, \omega, P)$, где V — алфавит (набор символов), $\omega \in V^+$ — непустое слово, которое мы будеми называть *аксиомой* и $P \subset V \times V^*$ — конечный набор *продукций*. Продукция вида (a, μ) записывается следующим образом: $a \to \mu$. Если для какого-то $a \in V$ не указано явно продукция, то считается, что существует продукция вида $a \to a$.

Определим теперь правила обработки продукций, или как их ещё называют, правила вывода.

Пусть $\mu = a_1 \dots a_m$ произвольное слово над алфавитом V. Слово $\nu = \chi_1 \dots \chi_m \in V^*$ напрямую выводится из (или порождается) словом μ (записывается $\mu \Rightarrow \chi$) в том и только в том случае, если $a_i \to \chi_i$ для всех $i = 1 \dots m$.

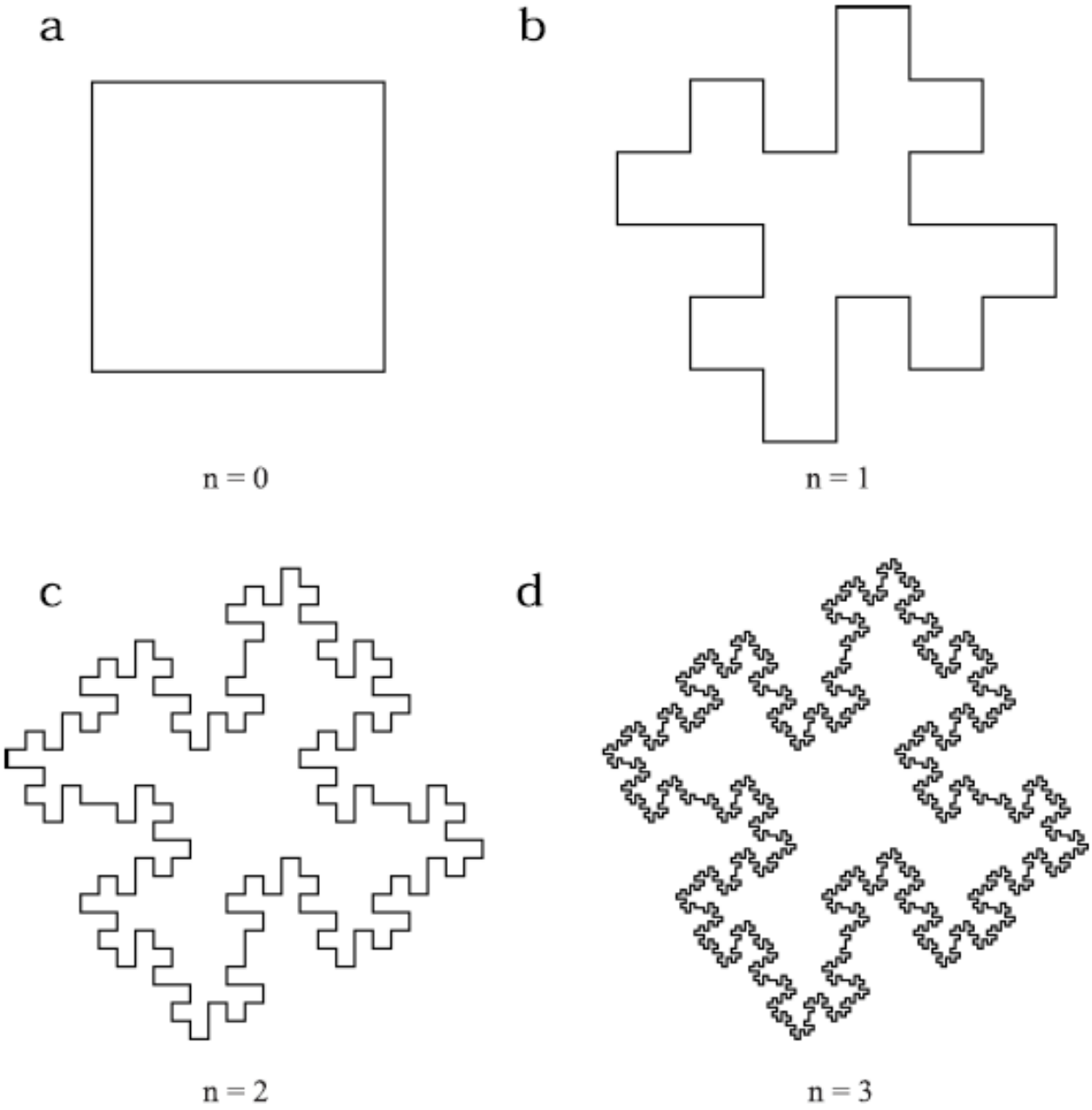
Слово ν порождается L-системой G, если существует последовательность слов $\mu_0, \mu_1, \dots,$ такая, что $\mu_0 = \omega, \mu_n = \nu$ и $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$

Рассмотрим простой пример L-системы, интерпретируемой черепашкой.

$$\omega : F - F - F - F$$

$$F \to F - F + F + FF - F - F + F$$

Порождаемые этими правилами картинки для $\mu_0 = \omega, \mu_1, \mu_2, \mu_3$ приведены ниже:



Здесь для получения μ_1 в строке с аксиомой ω были последовательно заменены все символы `F` на соответствующие продукции `F-F+F+FF-F-F+F`. Результат такой подстановки — строка `F-F+F+FF-F-F+F-F-F+F+FF-F-F+F-F-F+F+FF-F-F+F-F-F+F+FF-F-F+F`. Обратите внимание, что знаки `+` и `-` не интерпретируются алгебраически, это просто конкатенация строк.

L. Придумайте правила вывода для задач рисования фракталов из предыдущих пунктов и нарисуйте соответствующие фракталы при помощи черепашки.

До сих пор мы использовали единственное правило вывода. Попробуем использовать два:

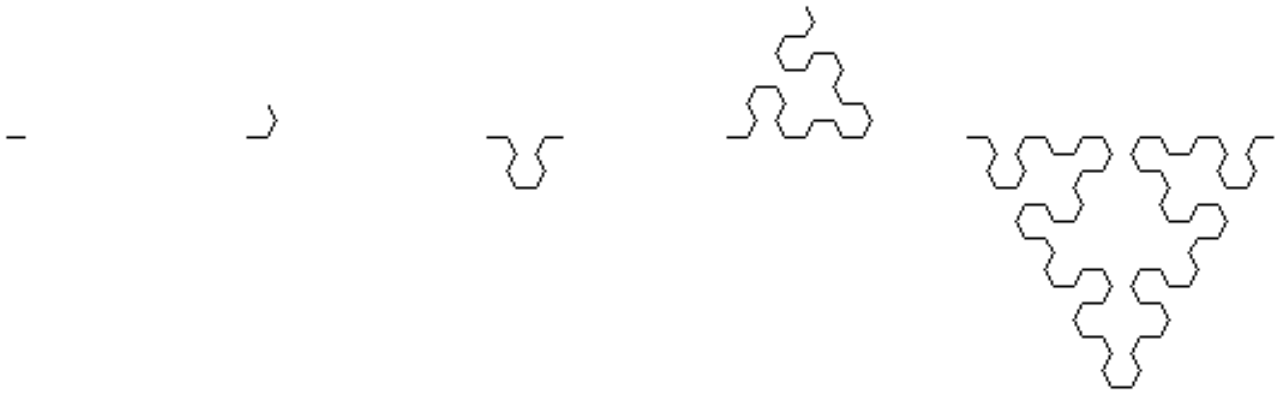
$$\omega : F_L$$
$$F_L \to F_L - F_R +$$
$$F_R \to F_R - F_L +$$

Ниже приведены результаты первых 6 итераций.

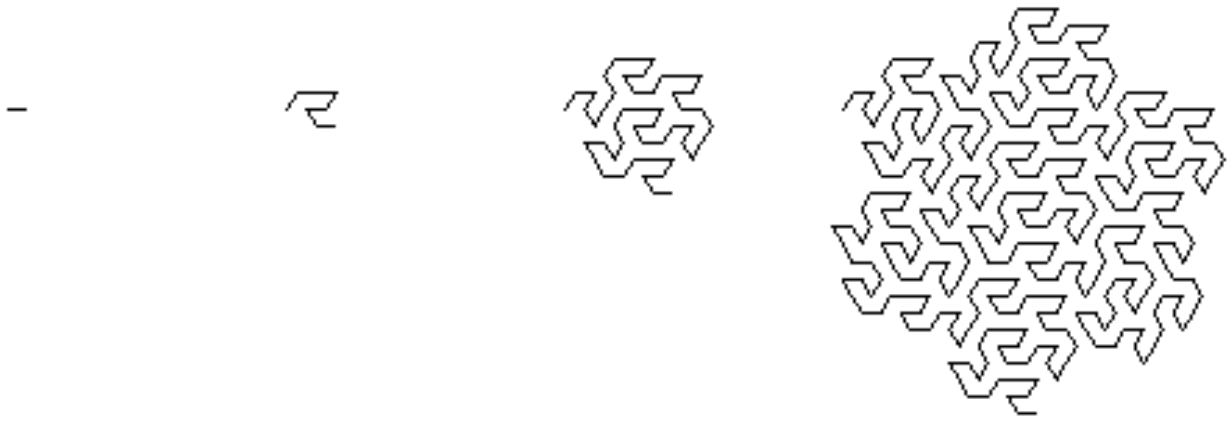
При реализации можно вместо F_L использовать L, а вместо F_R использовать R. Перед рисованием обе буквы заменить на F.

M. Используя два вида продукций, придумайте правила вывода кривой дракона, первые 6 итераций построения которой изображены на рисунке.

N. Используя два вида продукций, придумайте правила вывода "ковра Серпиньского", первые 5 итераций построения которого изображены на рисунке (угол поворота здесь равен 60^0).



О. Используя два вида продукций, придумайте правила вывода фрактала, первые 4 итерации построения которого изображены на рисунке (угол поворота здесь равен 60^0).



Добавим к языку, используемому черепашкой, ещё два символа — квадратные скобки: `[` и `]`. Дополним правила интерпретации следующим образом:

`[` Запишем в стек текущее состояние черепашки — координаты и направление (можно записывать и другие атрибуты, но мы пока ограничимся этим)

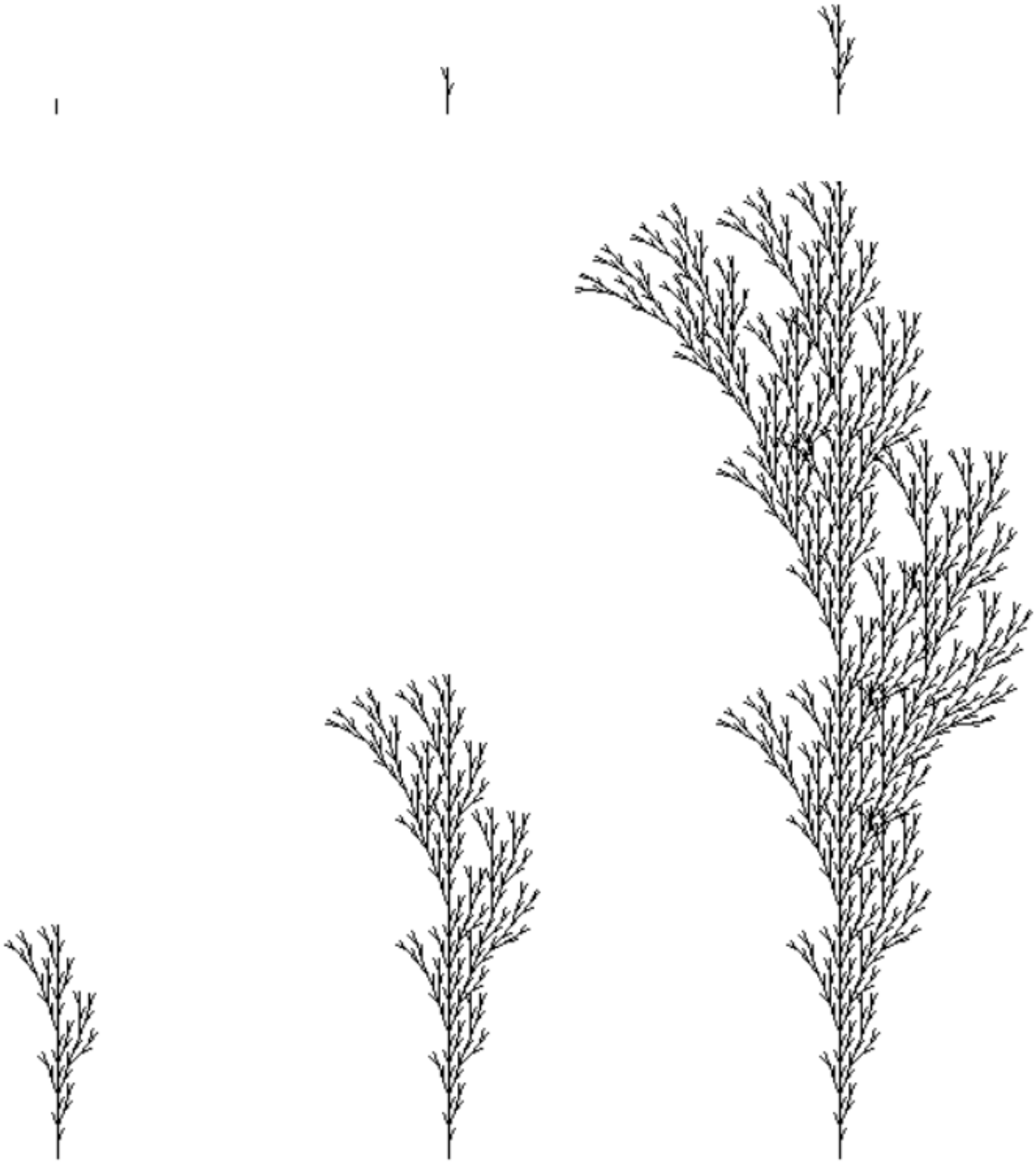
`]` Вынем с верхушки стека запомненные ранее атрибуты и сделаем их текущими атрибутами черепашки

Попробуйте получить при помощи следующих правил вывода приведённые ниже картинки.

$$\delta = 20^0$$

$$\omega : F$$

$$F \to F[+F]F[-F][F]$$



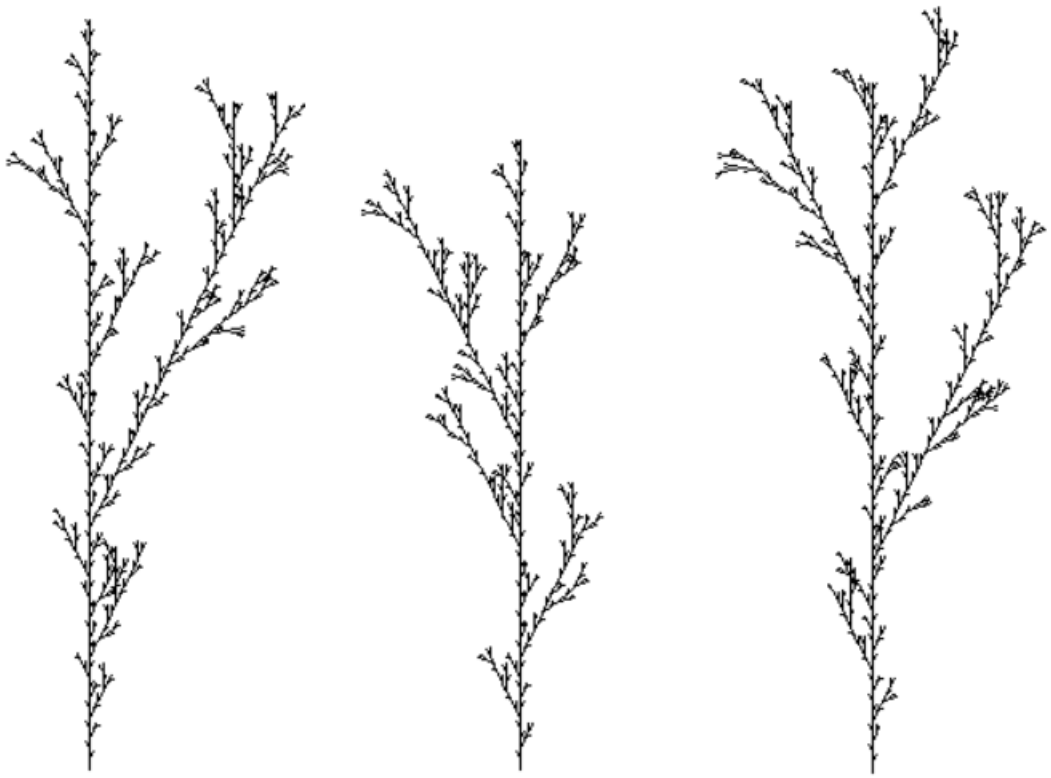
P. Кроме того, можно отказаться от *детерминированности* правил вывода и добавить в этот процесс случайность. А именно, пусть символу соответствует не одно правило вывода, а несколько. А то, какое именно правило применяется, определяется случайным образом. Например:

$$\delta = 25.7^0$$

$$\omega : F$$

$$\begin{aligned} F &\xrightarrow{1/3} F[+F]F[-F]F \\ F &\xrightarrow{1/3} F[+F]F \\ F &\xrightarrow{1/3} F[-F]F \end{aligned} \tag{1}$$

Ниже приведён пример запуска *одной и той же программы* с указанными правилами.



Справочник по модулю `turtle`

Прежде всего, ссылка на стандартную документацию.

Основное

- `T.forward(distance)` — проползти вперёд на distance пикселей
- `T.right(angle)` — повернуть направо (по часовой стрелке) на угол `angle` градусов
- `T.left(angle)` — повернуть налево (против часовой стрелки) на угол `angle` градусов
- `T.pendown()` — опустить перо (начать рисование)
- `T.penup()` — поднять перо (закончить рисование)
- `T.goto(x, y)` — переместить черепашку в точку с координатами (x, y)

Движение и направление

- `T.backward(distance)` — проползти назад на `distance` пикселей
- `T.setx(x)` — установить `x` координату черепашки
- `T.sety(y)` — установить `y` координату черепашки
- `T.setheading(to_angle)` — повернуть черепашку под углом `to_angle` к вертикали (0 — наверх, 90 — направо)
- `T.home()` — вернуть черепашку домой, в точку с координатами $(0, 0)$
- `T.dot(size, color)` — нарисовать точку диаметра `size` цвета `color`. Параметр `color` необязателен
- `T.undo()` — откатить предыдущее действие черепашки

Рисование

- `T.pensize(width)` — установить диаметр пера в `width`
- `T.pencolor(colorstring)` — установить цвет линии, которая рисует черепашка (например, `'brown'` или `'#32c18f'`)
- `T.fillcolor(colorstring)` — установить цвет заполнения
- `T.begin_fill()` — начать следить за черепашкой для заполнения области
- `T.end_fill()` — заполнить цветом `fillcolor` область, пройденную черепашкой начиная с `begin_fill`
- `T.showturtle()` — показать черепашку
- `T.hideturtle()` — спрятать черепашку
- `T.write(text)` — вывести текст `text`

Скорость

- `T.speed(speed)` — установить скорость черепашки. Значение `speed` должно быть от 1 (медленно) до 10 (быстро), или 0 (мгновенно)
- `T.getscreen().tracer(n)` — отрисовывать лишь каждый n-й кадр. Почти в n раз ускоряет рисование.

Информация о черепашке

- `T.position()` — получить текущие координаты черепашки
- `T.towards(x, y)` — получить угол между текущим направление черепашки и прямой от черепашки к точке (x, y)
- `T.xcor()` — получить x координату черепашки
- `T.ycor()` — получить y координату черепашки
- `T.heading()` — получить текущий угол к вертикали
- `T.distance(x, y)` — получить расстояние до точки (x, y)
- `T.isdown()` — узнать, рисует ли сейчас черепашка (опущено ли перо)
- `T.isvisible()` — узнать, видима ли сейчас черепашка

Пример программы

```
from turtle import *    # Подключаем модуль turtle
T = Turtle()            # T - это наша черепашка. Можно создавать много черепашек
T.pendown()             # Опускаем перо (начало рисования)
T.forward(50)           # Проползти 50 пикселей вперёд
T.left(90)              # Поворот влево на 90 градусов
T.forward(50)           # Рисуем вторую сторону квадрата
T.left(90)              # Поворот влево на 90 градусов
T.forward(50)           # Рисуем третью сторону квадрата
T.left(90)              # Поворот влево на 90 градусов
T.forward(50)           # Рисуем четвертую сторону квадрата
T.penup()               # Поднять перо (закончить рисовать)
T.forward(100)          # Отвести черепашку от рисунка в сторону
T.hideturtle()          # Спрятать черепашку
mainloop()              # Задержать окно на экране
```

Или то же самое, но с функцией и без намёка на использование нескольких черепашек

```
from turtle import *

def draw():
    pendown()
    forward(50)
    left(90)
    forward(50)
    left(90)
    forward(50)
    left(90)
    forward(50)
    penup()
    forward(100)
    hideturtle()

draw()
hideturtle()
mainloop()
```