# Фракталы

## Рекурсивный вызов функций

Вспомним рекурсивный вызов функции, когда функция вызывает саму себя.

### Пример 1: Рисуем дерево

Нарисуем дерево с помощью рекурсивного вызова. Пусть дерево каждый год вырастает на 1 уровень, каждая ветка дает 2 новые веточки, отклоняющиеся на угол +ang и -ang от направления роста ветки. Размер ветки size, а веточки меньше ее на d.

Функция **tree(size, d, ang, n)** будет рисовать дерево, которое росло n лет.

Функция tree нарисует ветку, вызовет саму себя 2 раза, чтобы нарисовать веточки, и вернется в точку, где начинала рисовать.

n - сколько лет дереву осталось расти.

|  |  |  |
| --- | --- | --- |
| tree(100, 20, 30, 1) | tree(100, 20, 30, 3) | tree(100, 20, 30, 5) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/e12_1_1.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/e12_1_3.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/e12_1_5.png |

# -\*- coding: utf-8 -\*-

import turtle

import time

'''

tree(size, d, ang, n) - рисовать дерево, которое росло n лет.

Пусть дерево каждый год вырастает на 1 уровень,

каждая ветка дает 2 новые веточки,

отклоняющиеся на угол +ang и -ang от направления роста ветки.

Размер ветки size, а веточки меньше ее на d.

'''

def tree(size, d, ang, n):

if n == 0: # дерево закончило расти, возвращаемся

return

t.fd(size) # рисуем ветку

t.left(ang) # поворачиваемся рисовать левую веточку

tree(size-d, d, ang, n-1) # рисуем левое поддерево (веточку и дальше)

# после этой функции черепаха будет в том же месте

# и повернута на тот же угол

t.left(-2\*ang) # поворачиваемся рисовать правую веточку

tree(size-d, d, ang, n-1) # рисуем правое поддерево (веточку и дальше)

# после этой функции черепаха будет в том же месте

t.left(ang) # возвращаем направление ветки

t.fd(-size) # возвращаемся в начало ветки

# конец функции

t = turtle.Turtle()

t.shape("turtle")

t.width(3)

t.speed(0)

t.seth(90)

t.up()

t.bk(200)

t.down()

t.color('brown')

tree(100, 20, 30, 1)

turtle.done()

### Задача 1: Нарисовать дерево с зелеными листьями

Ветки последнего года у дерева нарисовать зелеными (green). Старые ветки рисовать коричневыми (brown).

|  |  |  |
| --- | --- | --- |
| tree(100, 20, 30, 1) | tree(100, 20, 30, 3) | tree(100, 20, 30, 5) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/t12_1_1.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/t12_1_3.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/t12_1_5.png |

## Фракталы (теория)

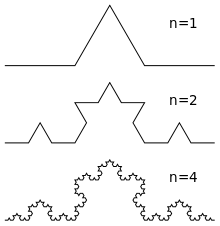
[Фракта́л](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B0%D0%BA%D1%82%D0%B0%D0%BB) (лат. fractus — дроблёный, сломанный, разбитый) — математическое множество, обладающее свойством самоподобия (объект, в точности или приближённо совпадающий с частью себя самого, то есть целое имеет ту же форму, что и одна или более частей)

Один из способов нарисовать фрактальную кривую - рисовать заменяя отрезок на набор отрезков.

[Список фракталов на английском языке](https://en.wikipedia.org/wiki/List_of_fractals_by_Hausdorff_dimension)

### Пример 2: Кривая Коха

[Кривая Коха](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%9A%D0%BE%D1%85%D0%B0) - на каждой итерации каждый отрезок заменяется на такой набор отрезков:



Функция **koch\_line(size, n)** рисует на отрезке длины size кривую Коха и делает n итераций (рисует кривую Коха глубины n)

|  |  |  |
| --- | --- | --- |
| koch\_line(200, 0) | koch\_line(200, 1) | koch\_line(200, 3) |
| http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/e12_2_0.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/e12_2_1.png | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractal/e12_2_3.png |

# -\*- coding: utf-8 -\*-

import turtle

import time

def koch\_line(size, n):

if n == 0: # рисуем линию и дальше не идем

t.fd(size)

return

a = size/3 # иначе разбиваем отрезок

# и вместо него делаем набор отрезков

koch\_line(a, n-1)

t.left(60)

koch\_line(a, n-1)

t.right(120)

koch\_line(a, n-1)

t.left(60)

koch\_line(a, n-1)

# конец функции

t = turtle.Turtle()

t.shape("turtle")

t.width(3)

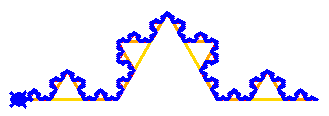
t.speed(0)

t.color('blue')

koch\_line(200, 3)

turtle.done()

Можем нарисовать итерацию одну за другой разным цветом:



# -\*- coding: utf-8 -\*-

import turtle

import time

tones = [

'yellow', # tones[0]

'gold', # tones[1]

'orange', # tones[2]

'red', # tones[3]

'violet', # tones[4]

'blue', # tones[5]

'green' # tones[6]

]

def koch\_line(size, n):

if n == 0: # рисуем линию и дальше не идем

t.fd(size)

return

a = size/3 # иначе разбиваем отрезок

# и вместо него делаем набор отрезков

koch\_line(a, n-1)

t.left(60)

koch\_line(a, n-1)

t.right(120)

koch\_line(a, n-1)

t.left(60)

koch\_line(a, n-1)

# конец функции

t = turtle.Turtle()

t.shape("turtle")

t.width(3)

t.speed(0)

for i in range(5): # i = 0..4

p0 = t.pos() # запомнили начальную позицию в p0

t.color(tones[i]) # новой итерации - новый цвет

koch\_line(300, i) # нарисовали кривую Коха глубины i

t.up() # вернулись на начальную позицию

t.goto(p0)

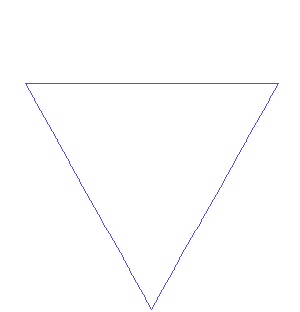
t.down()

time.sleep(1) # ждем 1 секунду

turtle.done()

### Задача 2: Снежинка Коха

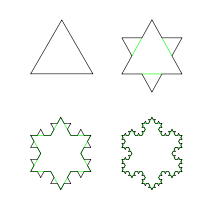
Снежинка Коха (при увеличении глубины):



Написать функцию **koch\_tri(size, n)**, которая рисует снежинку Коха глубины n, у которой сторона первого треугольника size.

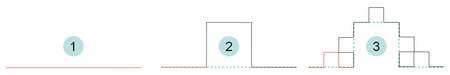
Результаты вызовов функции:

* koch\_tri(200, 0)
* koch\_tri(200, 1)
* koch\_tri(200, 2)
* koch\_tri(200, 3)



### Задача 3а: Построить кубическую кривую 1 типа

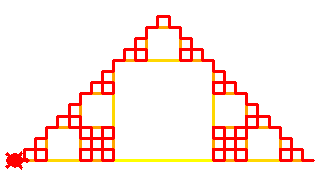
Итерации разной глубины:



Конечный результат:

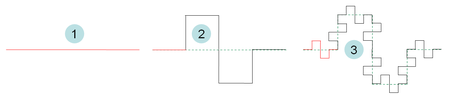


Вызов функции для разных глубин и цветов:



### Задача 3b: Построить кубическую кривую 2 типа (кривая Минковского)

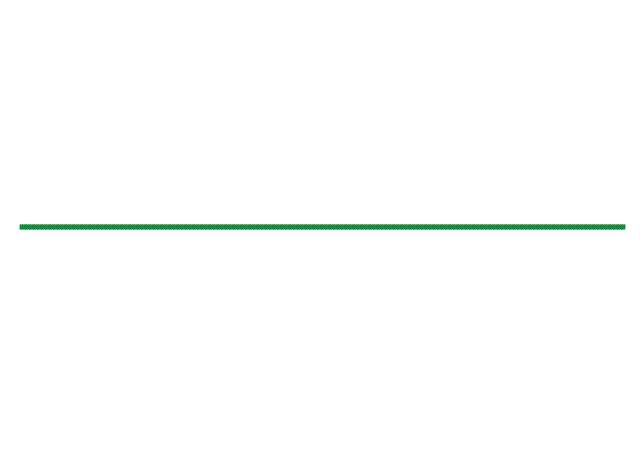
Итерации разной глубины:



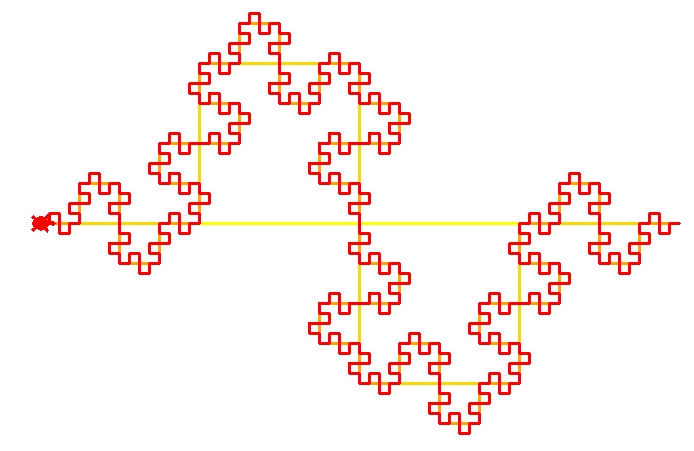
Конечный результат:



Еще одна картинка, где показывают как получается кривая:



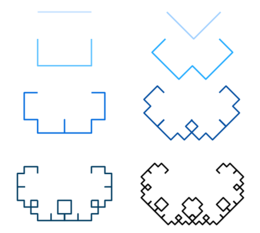
Вызов функции для разных глубин и цветов:

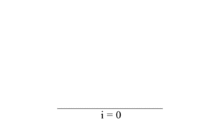


### Задача 3c: Кривая Леви

Написать функцию **levy\_line(size, n)**, которая рисует [кривую Леви](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%9B%D0%B5%D0%B2%D0%B8) глубины n, при длине отрезка size.

При увеличении глубины форма кривой меняется так:

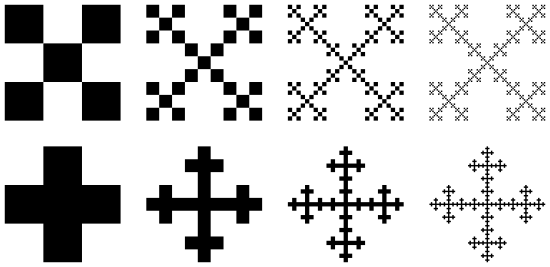




|  |  |
| --- | --- |
| koch\_line(320, 4) | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractalLevi/t12_levi_4.png |
| koch\_line(320, 7) | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractalLevi/t12_levi_7.png |
| koch\_line(320, 10) | http://acm.mipt.ru/twiki/pub/Cintro/PythonFractalLevi/t12_levi_10.png |

## Кресты Висека

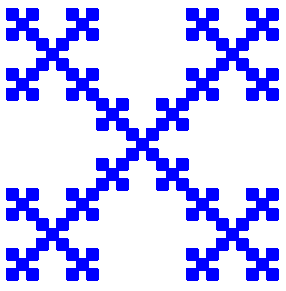
[Теория](https://en.wikipedia.org/wiki/Vicsek_fractal)



### Задача 4a: Крест-1

Написать функцию, которая рисует крест глубины n.

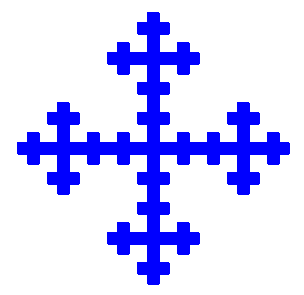
Пример вызова функции для глубины 3



### Задача 4b: Крест-2

Написать функцию, которая рисует крест глубины n.

Пример вызова функции для глубины 3



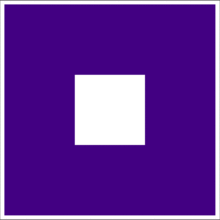
## Ковер и треугольник Серпинского

### Задача 4c: Ковер Серпинского

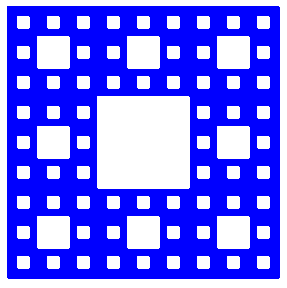
[Ковер Серпинского](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B2%D1%91%D1%80_%D0%A1%D0%B5%D1%80%D0%BF%D0%B8%D0%BD%D1%81%D0%BA%D0%BE%D0%B3%D0%BE)

Написать функцию, которая рисует ковер Серпинского глубины n.

Ковер Серпинского с увеличением глубины выглядит так:



Ковер Серпинского глубины 3:



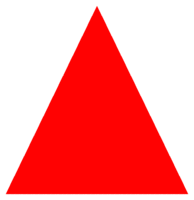
### Задача 4d: Треугольник Серпинского

[Треугольник Серпинского](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D1%83%D0%B3%D0%BE%D0%3Cbr%20/%3E%E2%80%A2D1%8C%D0%BD%D0%B8%D0%BA_%D0%A1%D0%B5%D1%80%D0%BF%D0%B8%D0%BD%D1%81%D0%BA%D0%BE%D0%B3%D0%BE)

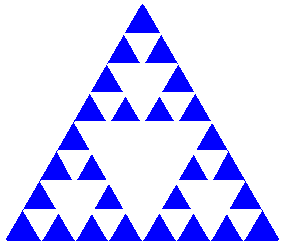
Написать функцию, которая рисует треугольник Серпинского глубины n.

Треугольник Серпинского с увеличением глубины выглядит так:

ierpinsky_triangle.png



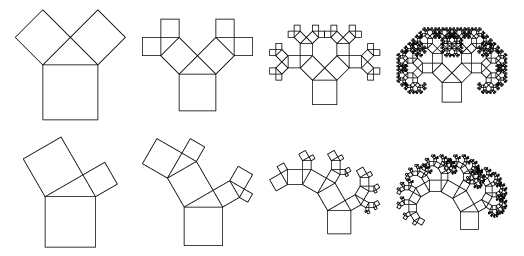
Треугольник Серпинского глубины 3:



## Дерево Пифагора

[Дерево Пифагора](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%9F%D0%B8%D1%84%D0%B0%D0%B3%D0%BE%D1%80%D0%B0) основано на построении "Пифагоровы штаны", которое используется для доказательства теоремы Пифагора (квадрат гипотенузы равен сумме квадратов катетов).

Классическое дерево строится на прямоугольном треугольнике с углами в 45o. Можно построить дерево на непрямоугольном треугольнике. Или взять не равнобедренный треугольник ("дерево, обдуваемое ветром").

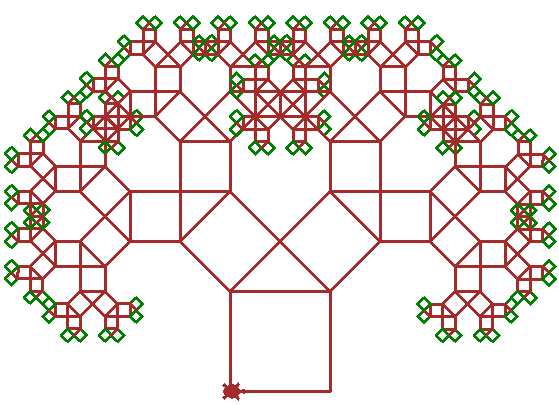


### Задача 5а: Классическое дерево Пифагора

Написать функцию, которая строит дерево Пифагора глубины n.

На квадрате построен равнобедренный треугольник с углом при основании 45o.

Построенное дерево глубины 7.

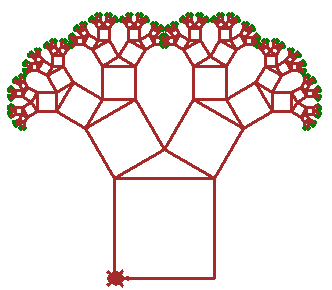


### Задача 5b: Дерево Пифагора с равнобедренным треугольником и углом при основании 30o

Написать функцию, которая строит дерево Пифагора глубины n.

На квадрате построен равнобедренный треугольник с углом при основании 30o.

Построенное дерево глубины 7.

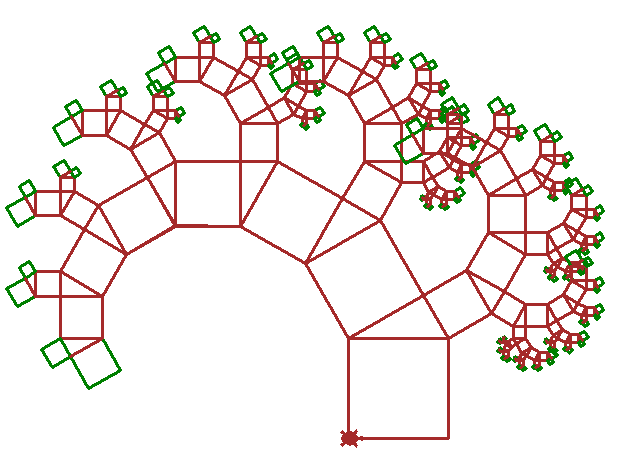


### Задача 5c: Дерево Пифагора, обдуваемое ветром

Написать функцию, которая строит дерево Пифагора глубины n.

На квадрате построен прямоугольный треугольник с углом при основании 30o.

Построенное дерево глубины 7.

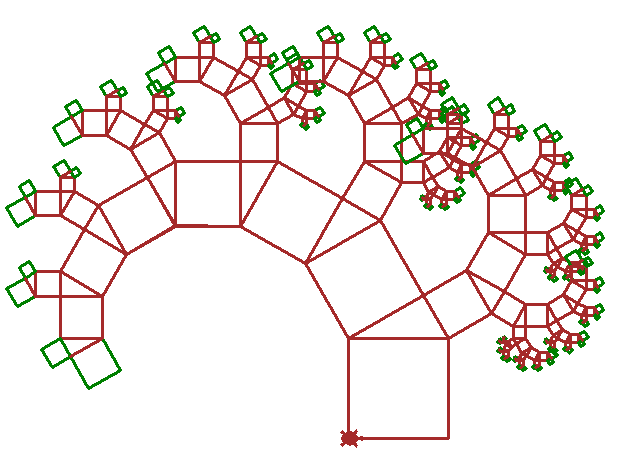


### Задача 5d: Обобщенное дерево Пифагора, обдуваемое ветром

Написать функцию, которая строит дерево Пифагора глубины n.

На квадрате построен прямоугольный треугольник с углом при основании ang градусов.

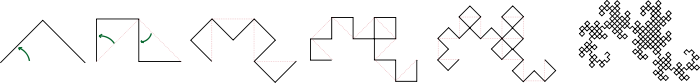
Построенное дерево для угла ang=30 и глубины 7.



## Создание последовательностей

### Пример: Кривая дракона (генерация последовательностей)

Написать функцию, которая строит [кривую дракона](https://en.wikipedia.org/wiki/Dragon_curve) глубины n.



Можно подойти к построению кривой по-другому. Запишем правила создания кривой как "что-то заменить на другое"

* Обозначим:
  + Первый отрезок обозначим как АF.
  + F - рисуем отрезок
  + + - повернуть направо на 90o
  + - - повернуть налево на 90o
  + A и B - их не рисуем, они нужны, чтобы описывать правила
* Напишем правила:
  + из А получаем A+BF+
  + из В получаем −FA−B

# -\*- coding: utf-8 -\*-

import turtle

import time

import math

def gen(size, n): # Первый отрезок обозначим как АF

t.fd(size)

A(size, n-1)

def A(size, n):

if n==0: # A не рисуем никак

return

# из А получаем A+BF+

A(size, n-1) # A

t.right(90) # +

B(size, n-1) # B

t.fd(size) # F

t.right(90) # +

def B(size, n):

if n==0: # B не рисуем никак

return

# из В получаем &#8722;FA&#8722;B

t.left(90) # -

t.fd(size) # F

A(size, n-1) # A

t.left(90) # -

B(size, n-1) # B

t = turtle.Turtle()

t.shape("turtle")

t.width(3)

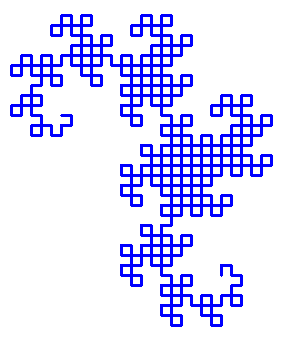
t.speed(0)

t.color('blue')

gen(10, 10) # правила применили глубиной 10

t.hideturtle()

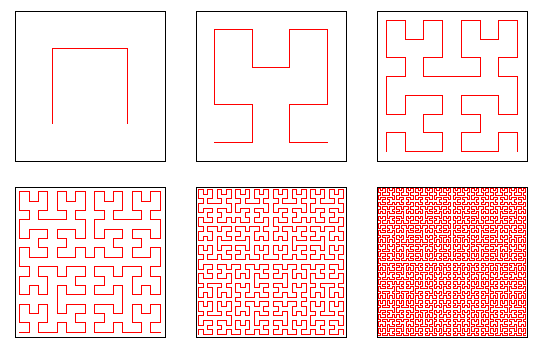
turtle.done()

* e12\_dragon.png:   
  

### Задача 6. Кривая Гильберта

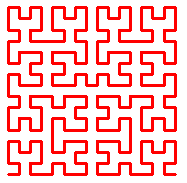
Написать функцию, которая строит [кривую Гильберта](https://en.wikipedia.org/wiki/Hilbert_curve) глубины n.

Кривую Гильберта можно задать разными способами:



Или можно записать правила:

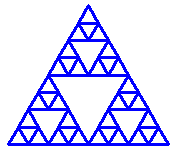
* Обозначим:
  + Первый отрезок обозначим как А
  + F - рисуем отрезок
  + + - повернуть направо на 90o
  + - - повернуть налево на 90o
  + A и B - их не рисуем, они нужны, чтобы описывать правила
* Напишем правила:
  + из А получаем - B F + A F A + F B -
  + из В получаем + A F - B F B - F A +



### Задача 7: Треугольник Серпинского через порождающие правила

Треугольник Серпинского можно задать через [порождающие правила:](https://en.wikipedia.org/wiki/L-system)

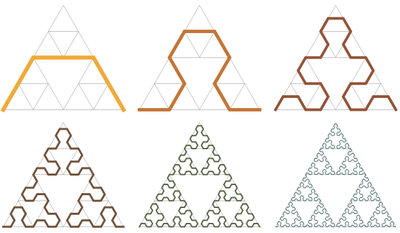
* Обозначим:
  + F, G - отрезок (рисуем вперед)
  + + - повернуть налево на 120 градусов
  + - - повернуть направо на 120 градусов
  + начинаем с F−G−G
* Правила:
  + из F получается F−G+F+G−F
  + из G получается GG

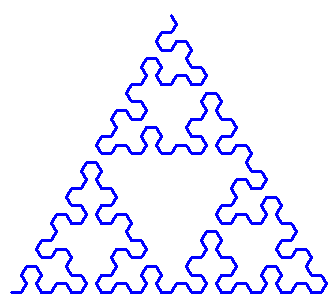


### Задача 8: Наконечник Серпинского

Треугольник Серпинского можно задать через [порождающие правила:](https://en.wikipedia.org/wiki/L-system)

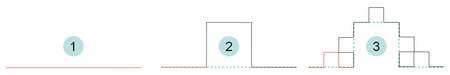
* Обозначим:
  + A, B - отрезок (рисуем вперед)
  + + - повернуть налево на 60 градусов
  + - - повернуть направо на 60 градусов
  + начинаем с A
* Правила:
  + из A получается B-A-B
  + из B получается A+B+A





## Задача:

**Напишите для кривой правила создания** и напишите функцию через эти правила. Кривая:



## В разработке

### Крест (3 варианта)

