Rule 30: chaotic!

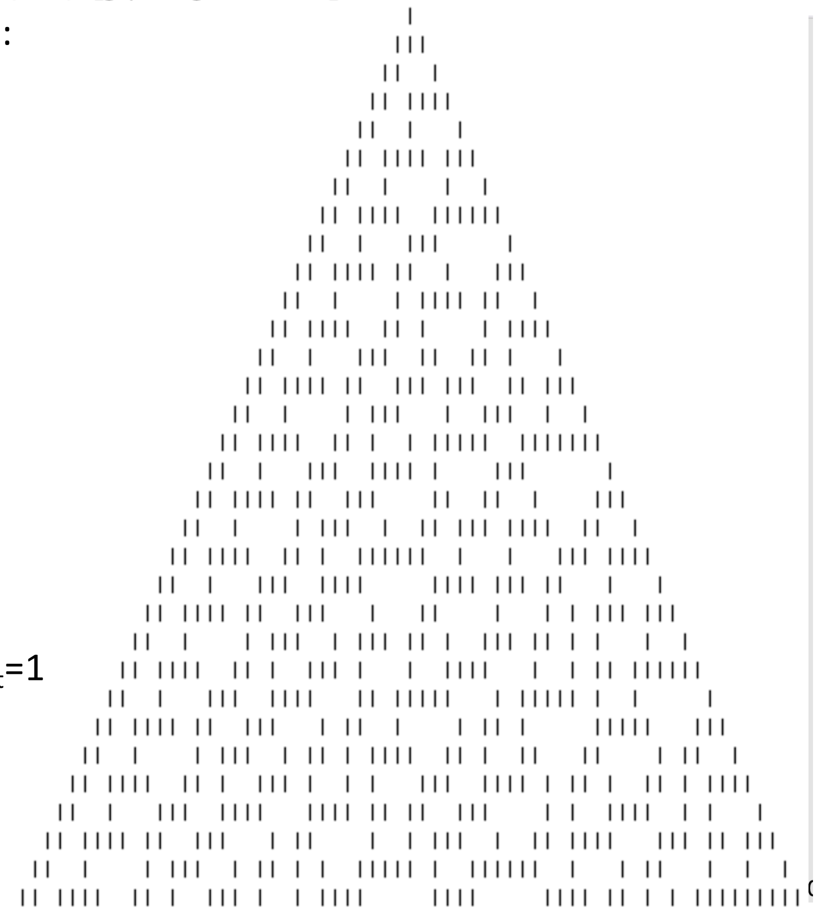
wiki

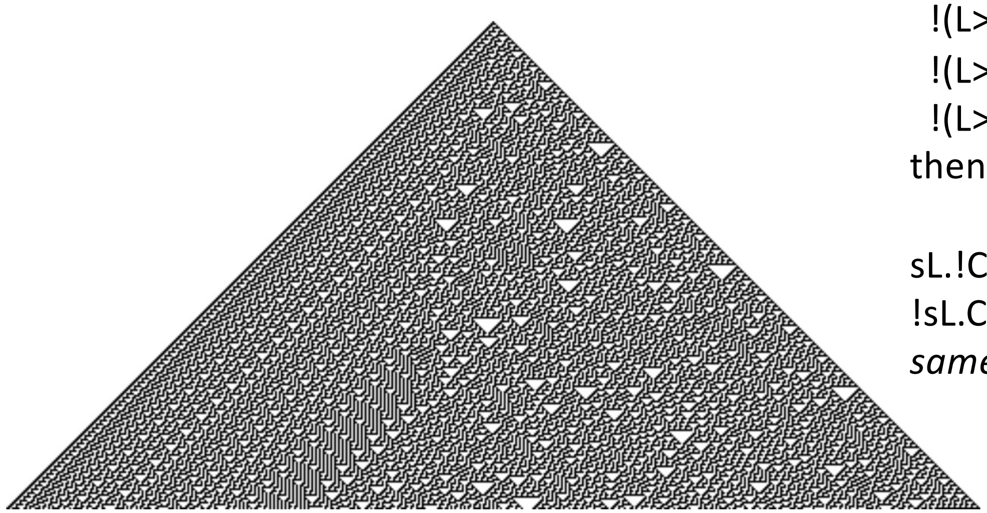
```
#include <stdint.h>
#include <iostream>
int main() {
 uint64_t state = 1u << 31;
 for (int i = 0; i < 32; ++i) {
  for (int j = 64; j--;)
   std::cout << char(state >> j & 1 ? '|' : ' ');
  std::cout << '\n';
  state = (state >> 1) ^ (state | state << 1);
 }
}
```

current pattern LCR	**111**	**110**	**101**	**100**	**011**	**010**	**001**	**000**
new state for center cell	0	0	0	1	1	1	1	0

```
if LCR in [100,011,010,001]:
 next C = 1
else:
 next C = 0

if
  (L>>1). !C. !(R<<1) or
 !(L>>1).  C.   (R<<1)  or
 !(L>>1).  C. !(R<<1) or
 !(L>>1). !C.  (R<<1)
then next C = 1
```

sL.!C.!sR + !sL.C.sR +
!sL.C.!sR + !sL.!C.sR => C_{next}=1
same as sL xor (C+sR) = 1

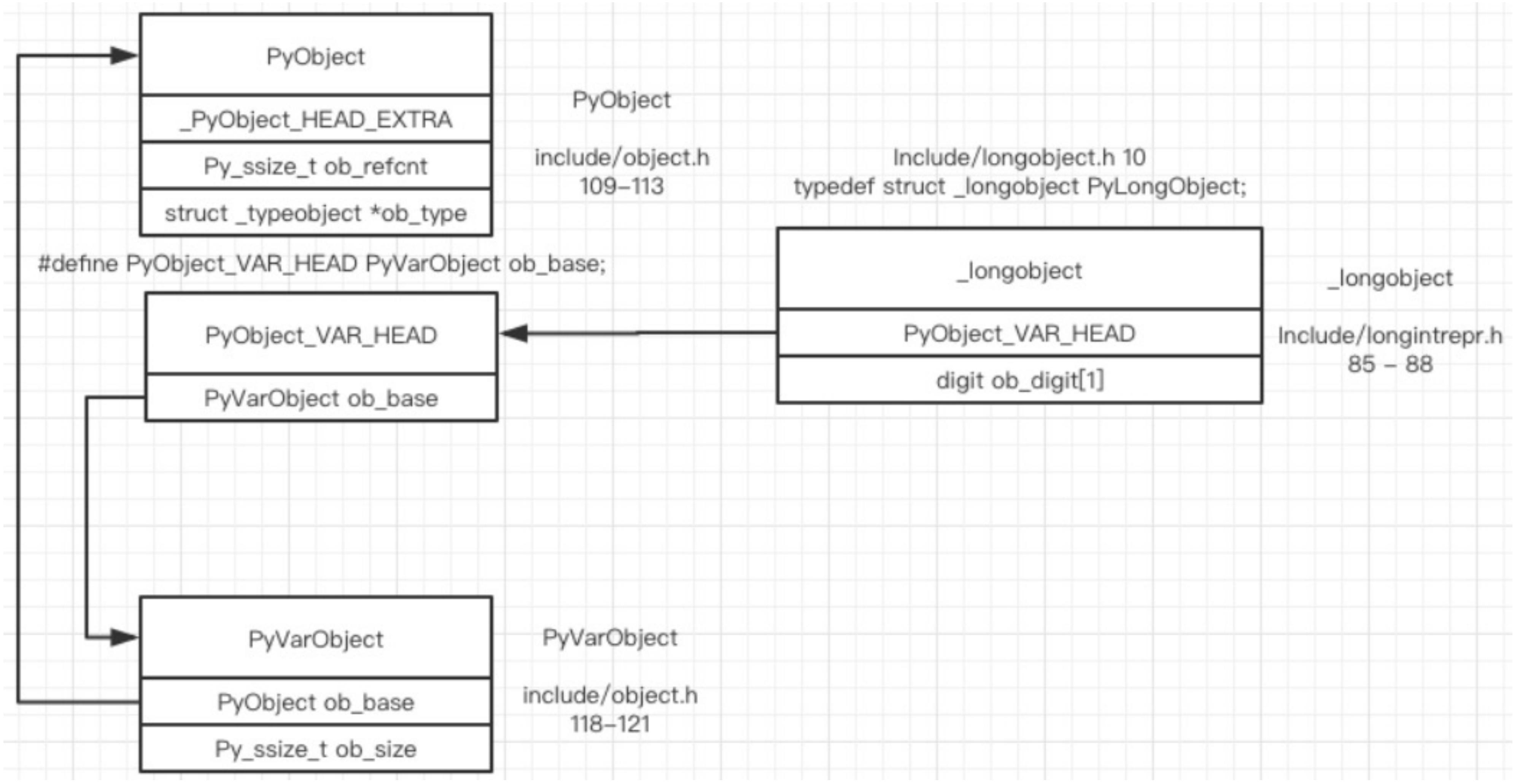




Rule 30 cellular automaton

Int in CPython

https://github.com/zpoint/CPython-Internals/blob/master/BasicObject/long/long.md

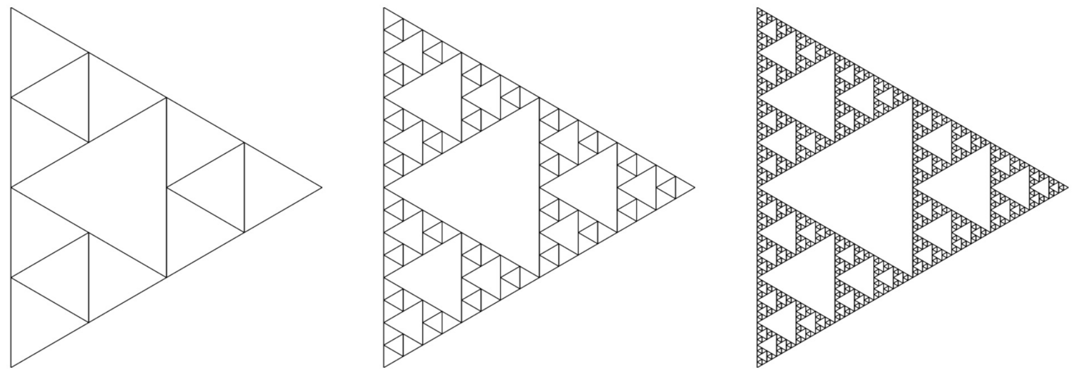


More details:

- 0: **ob_size = 0** indicates that **long object** represents integer 0
  - **ob_digit (16 bits)** field “don’t care”
- 1: **ob_size =** 1 and field in **ob_digit** = 1 with type **unsigned short**
- -1: **ob_size** = -1 and **ob_digit** =1
- 1023: **ob_size** = 1 and **ob_digit** = 1023. Same with 32767 (2^15 -1)
- 32768: **ob_size** = 2 and now **ob_digit** [2] with
  - ob_digit[0]=0
  - ob_digit[1]=1
- 262143 (2^18 -1): **ob_size** = 2 and now **ob_digit** [2] with
  - ob_digit[0]= 7FFF
  - ob_digit[1]= 7

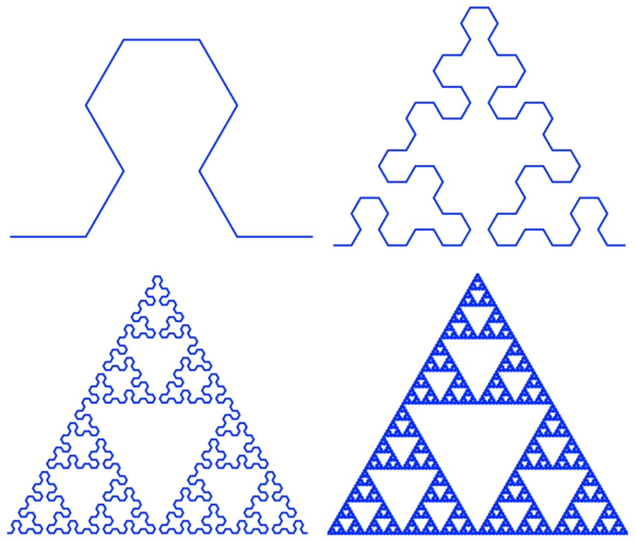
Note: leftmost bit in digit (bit 15)

reserved to handle carry. As integers become large, memory alloc as necessary: eg. adding 1 to 2^30 -1


n = 2 n = 4 n = 6

The Sierpinski triangle
drawn using an L-system.

- **variables** : F G
- **constants** : + −
- **start** : F−G−G
- **rules** : (F → F−G+F+G−F), (G → GG)
- **angle** : 120°
- F means "draw forward",
- G means "draw forward",
- + means "turn left by angle",
- − means "turn right by angle".


Evolution for *n* = 2, *n* = 4, *n* = 6, *n* = 8

Sierpiński arrowhead curve L-system.
variables : A B
constants : + −
start : A
rules : (A → B−A−B), (B → A+B+A)
angle : 60°
A and B both mean "draw forward",
+ means "turn left by angle", and
− means "turn right by angle" (see turtle graphics).

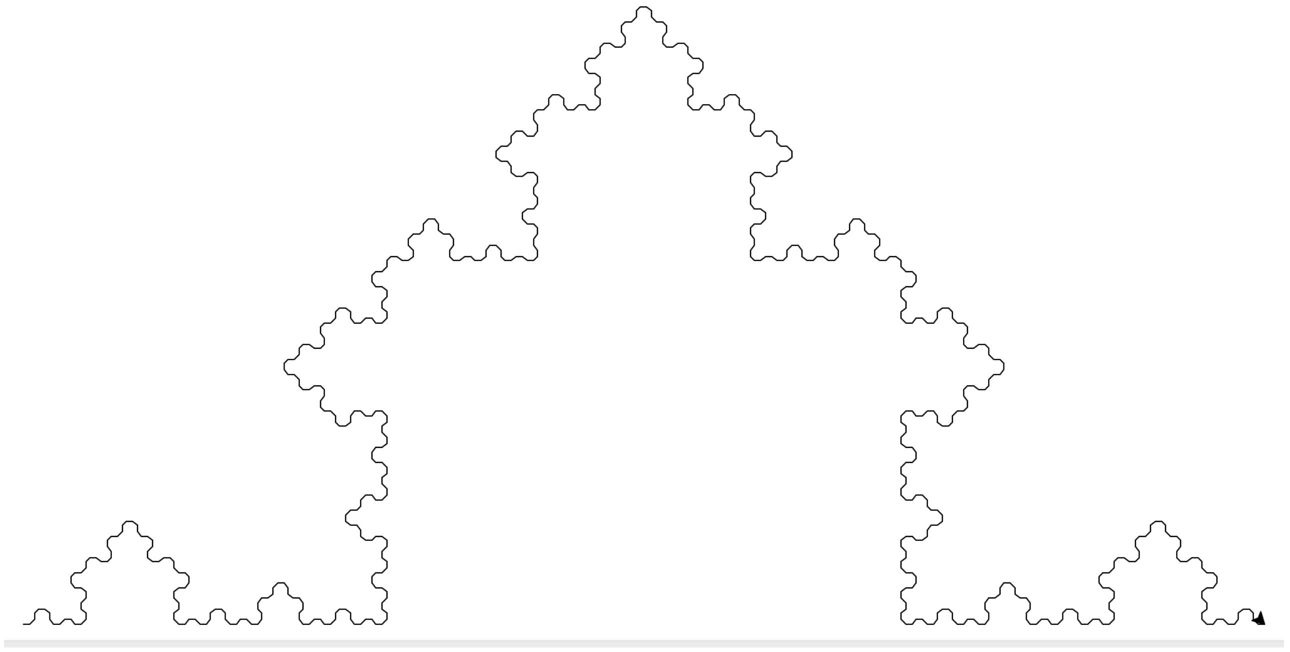
Sierpinski thru L

wiki

Sierpinski

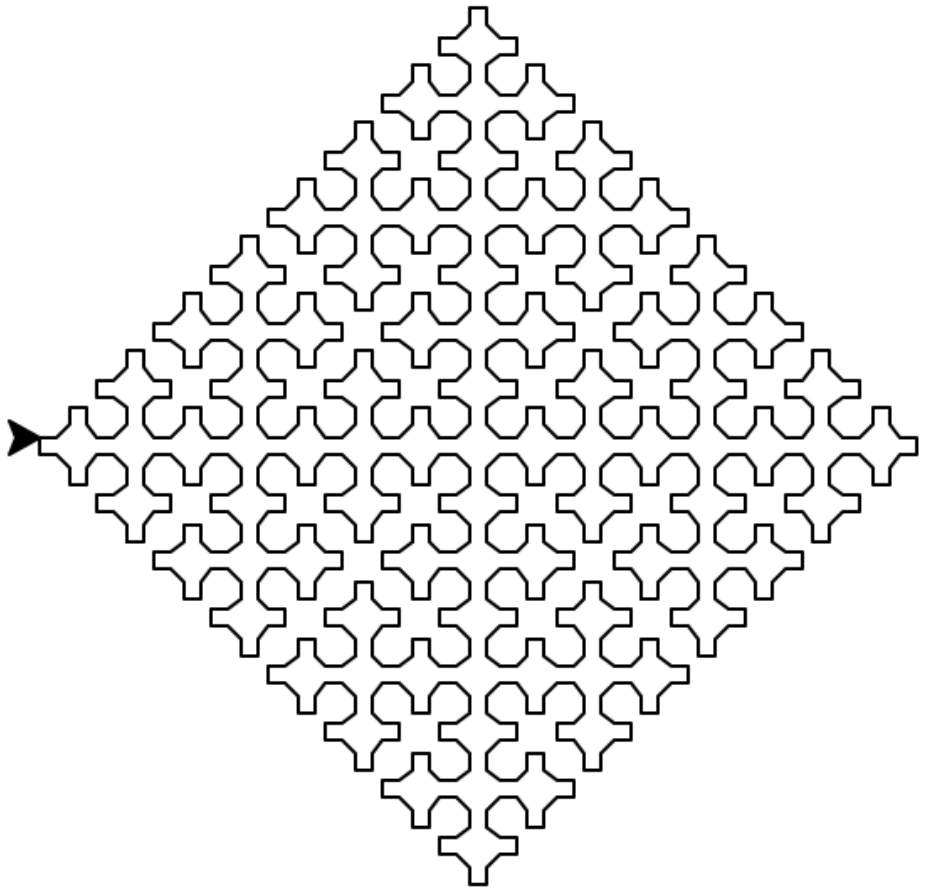
```
import turtle

# F = + R - F - R
# F = f
# R = - F + R + F
# R = r
pF = ["+", "R", "-", "F", "-", "R"]
pR = ["-", "F", "+", "R", "+", "F"]

f=5
r=5

def step(p):
 np=[]
 for i in range (0,len(p)):
   if (p[i]=="+"):
     np+=p[i]
   elif (p[i]=="-"):
     np+=p[i]
   elif (p[i]=="R"):
     np+=pR
     np+="-"
   elif (p[i]=="F"):
     np+=pF
     np+="+"
   else:
     np+=step(p[i])
 return(np)
```

```
def draw(p):
 for i in range (0,len(p)):
   if (p[i]=="+"):
     o.left(45)
   elif (p[i]=="-"):
     o.right(45)
   elif (p[i]=="R"):
     o.forward(r)
   elif (p[i]=="F"):
     o.forward(f)
   else:
     draw(p[i])

# main
o=turtle.Turtle()
p=pF
np=[]
for i in range(0,5):
  np=step(p)
  p=np
draw(p)
```



```
import turtle

pF = ["F","+","R","+","F","-","-","R","-","-","F","+","R","+","F"]
pR = "R”
p=["F","-","-","R","-","-","F","-","-","R","-","-"]

f=10
r=10

def step(p):
 np=[]
 for i in range (0,len(p)):
   if (p[i]=="+"):
     np+=p[i]
   elif (p[i]=="-"):
     np+=p[i]
   elif (p[i]=="R"):
     np+=pR
   elif (p[i]=="F"):
     np+=pF
   else:
     np+=step(p[i])
 return(np)
```

Pāmbu Kolam “Snake”



```
def draw(p):
 for i in range (0,len(p)):
   if (p[i]=="+"):
     o.left(45)
   elif (p[i]=="-"):
     o.right(45)
   elif (p[i]=="R"):
     o.forward(r)
   elif (p[i]=="F"):
     o.forward(f)
   else:
     draw(p[i])

o=turtle.Turtle()

np=[]
for i in range(0,4):
  np=step(p)
  p=np
draw(p)
```