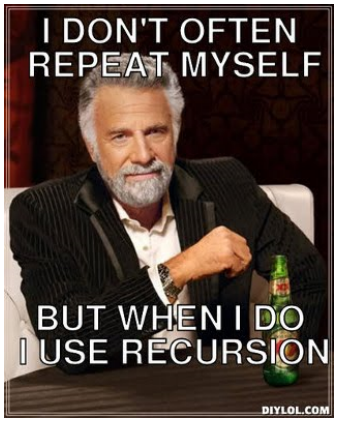
More Recursion



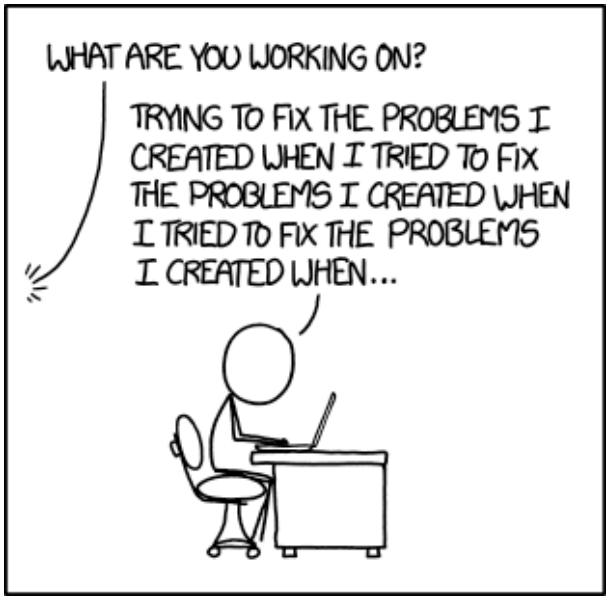
In order to understand recursion,
you must first understand recursion.

Let recursion do the work for you.

Exploit self-similarity

Produce short, elegant code

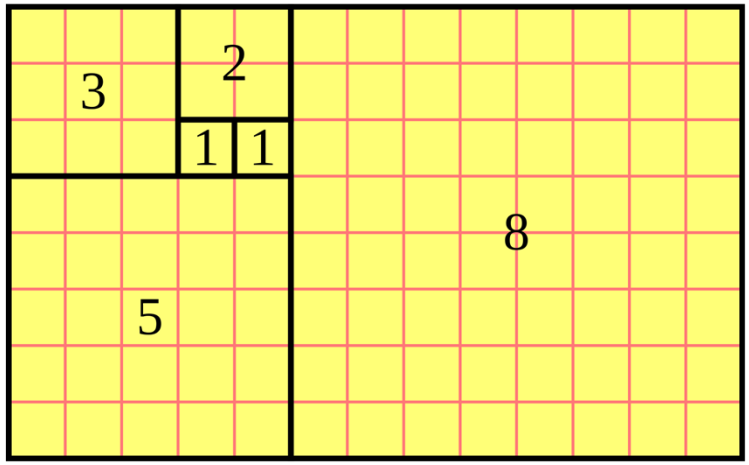
} **Less work !**



Fibonacci

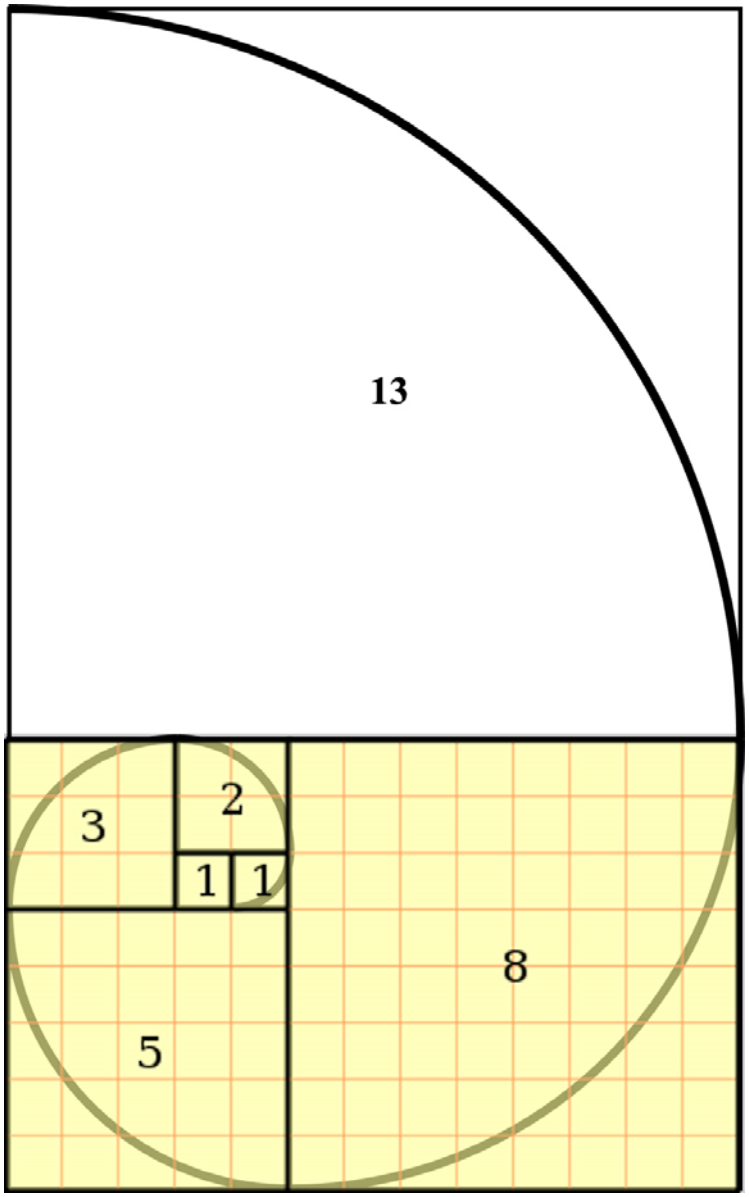
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ...

$F_n = F_{n-2} + F_{n-1}$

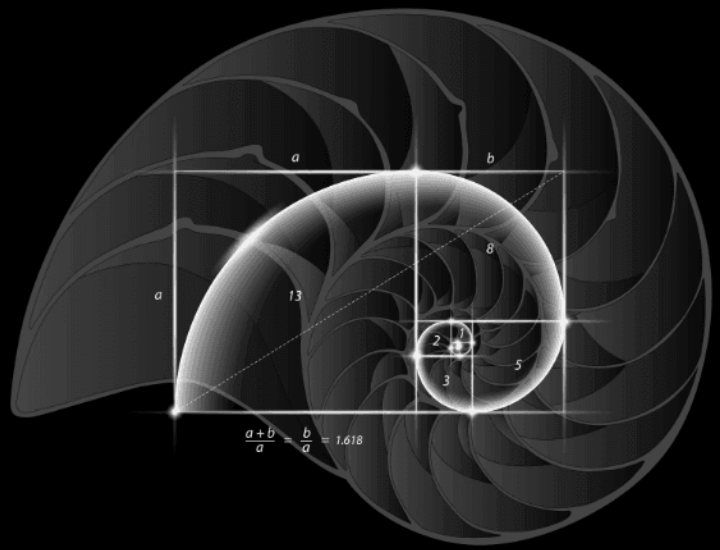


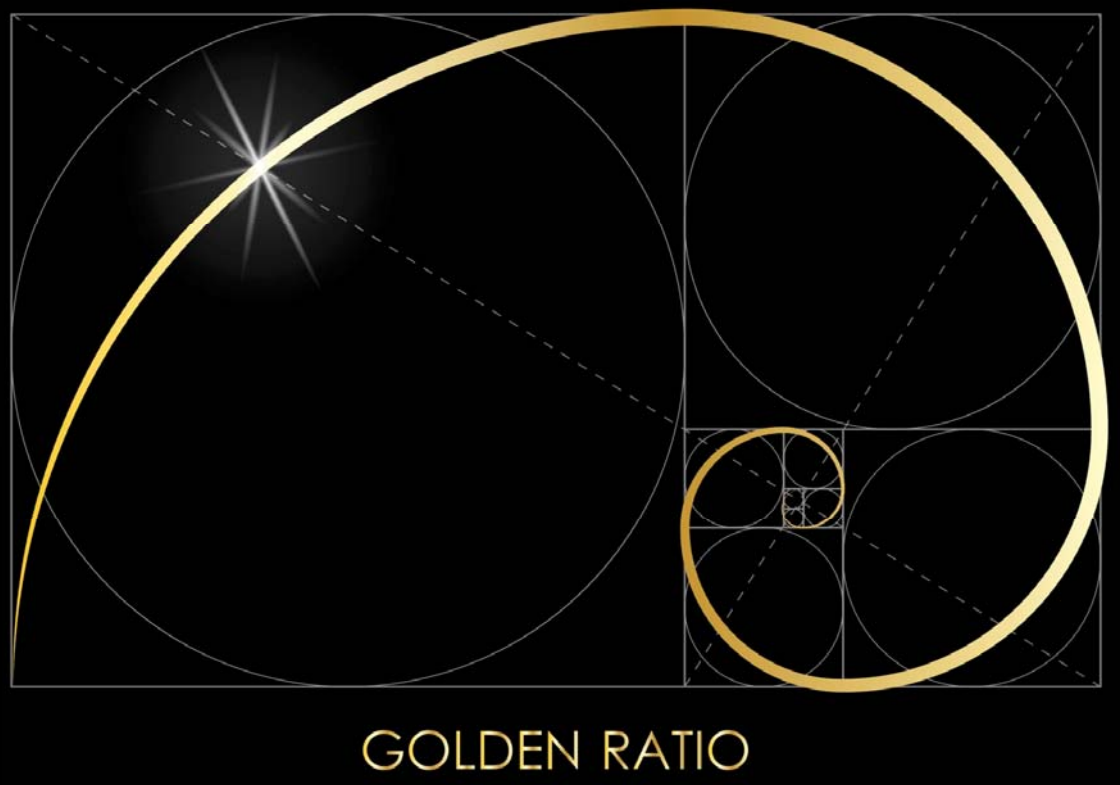
Fibonacci

$$F_n = F_{n-2} + F_{n-1}$$



F_n / F_{n-1} -> golden ratio





Fibonacci

```
# Returns Fibonacci number for N = 46
def fib(N):
```

fib.py

Fibonacci

```
# Returns Fibonacci number for any N ≤ 45
def fib_helper(N):
    …
```

```
# Returns Fibonacci number for N = 46
def fib(N):
```

fib.py

Fibonacci

```
# Returns Fibonacci number for any N ≤ 45
def fib_helper(N):
    …
```

```
# Returns Fibonacci number for N = 46
def fib(N):



        return fib_helper(N-1)+fib_helper(N-2)
```

fib.py

Fibonacci

```
# Returns Fibonacci number for any N ≤ 45
def fib_helper(N):
    …
```

```
# Returns Fibonacci number for any N ≤ 46
def fib(N):



        return fib_helper(N-1)+fib_helper(N-2)
```

fib.py

Fibonacci

```
# Returns Fibonacci number for any N ≤ 45
def fib_helper(N):
    …
```

```
# Returns Fibonacci number for any N ≤ 46
def fib(N):
    if N == 0:
        return 0
    elif N == 1:
        return 1
    else:
        return fib_helper(N-1)+fib_helper(N-2)
```

fib.py

Fibonacci

```
# Returns Fibonacci number for any N ≤ 45
def fib_helper(N):
    …
```

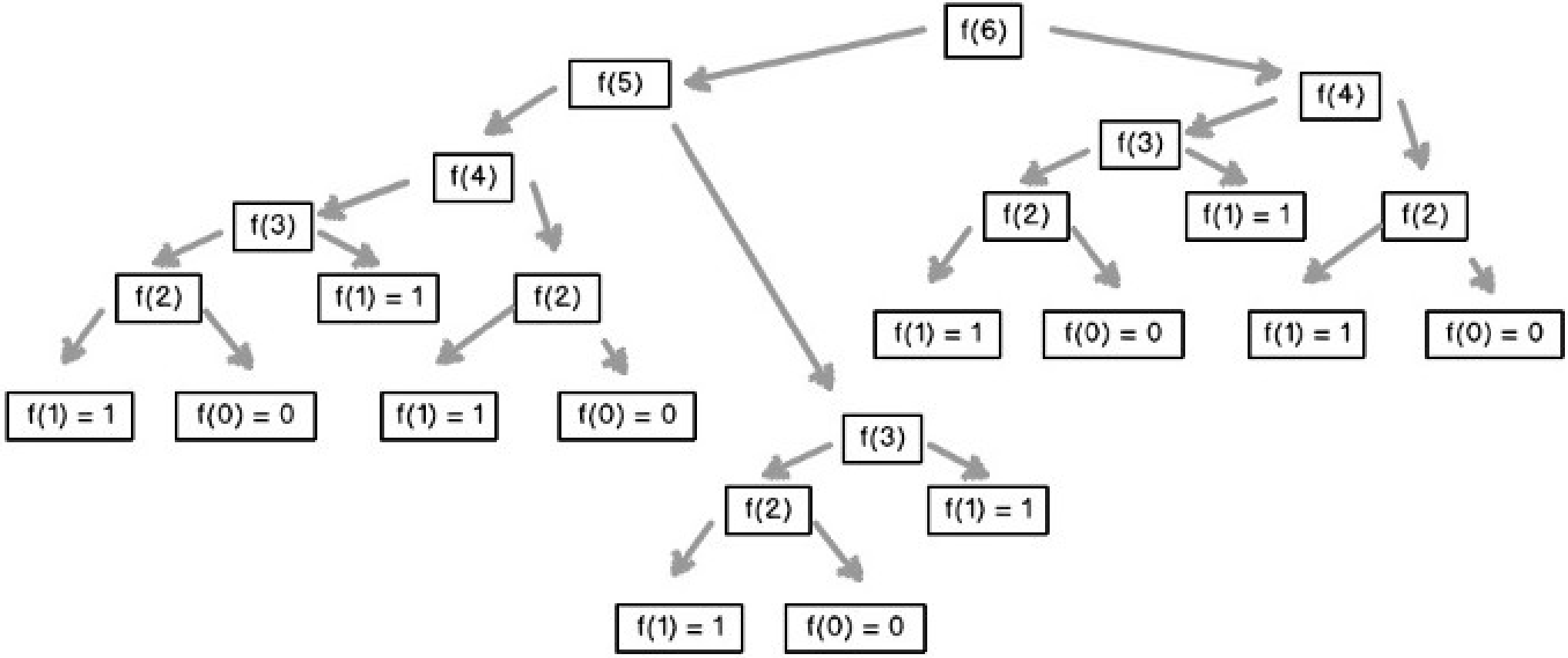
```
# Returns Fibonacci number for any N ≤ 46
def fib(N):
    if N == 0:
        return 0
    elif N == 1:
        return 1
    else:
        return fib(N-1)+fib(N-2)
```

fib.py

Fibonacci

```
# Returns Fibonacci number for any N ≤ 46
def fib(N):
    if N == 0:
        return 0
    elif N == 1:
        return 1
    else:
        return fib(N-1)+fib(N-2)
```

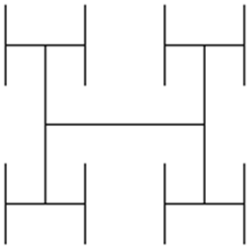
fib.py



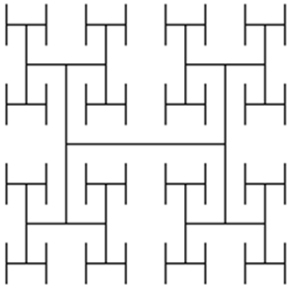
fib2.py fib_time.py

Graphical Recursion Example

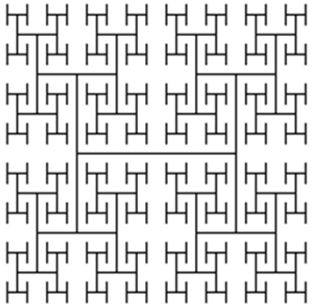
N = 1



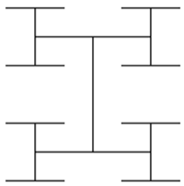
N = 2

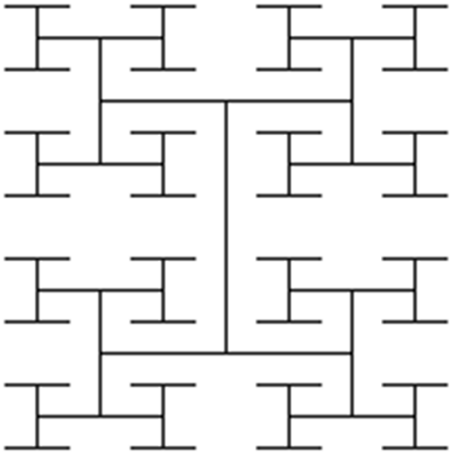


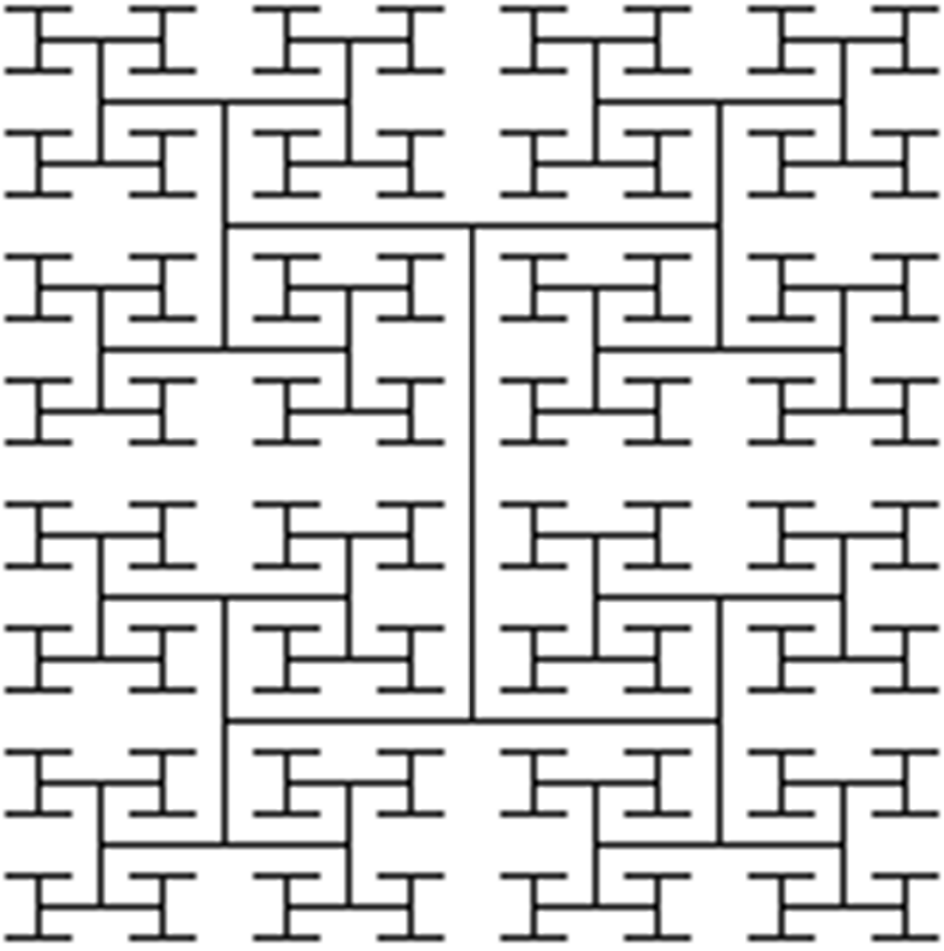
N = 3

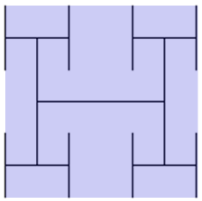


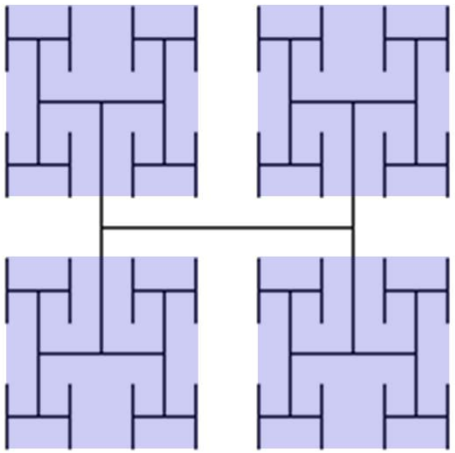
N = 4

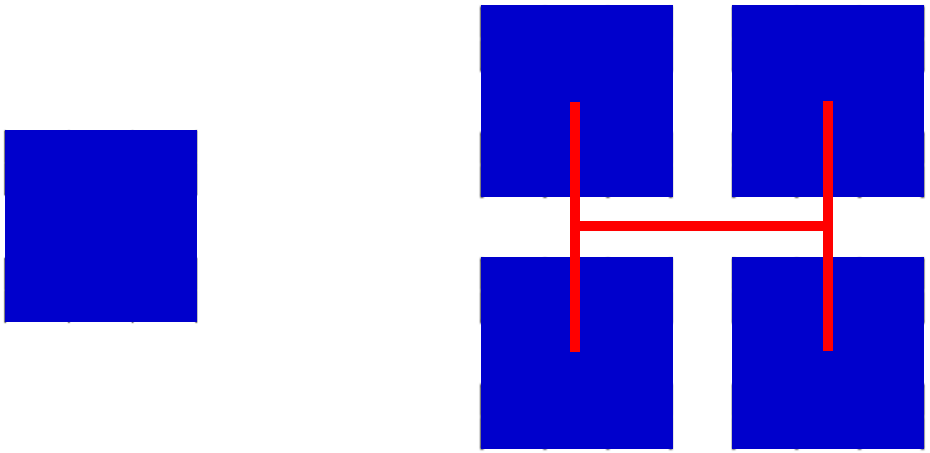




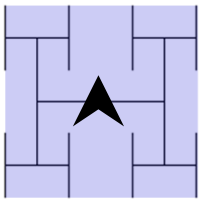


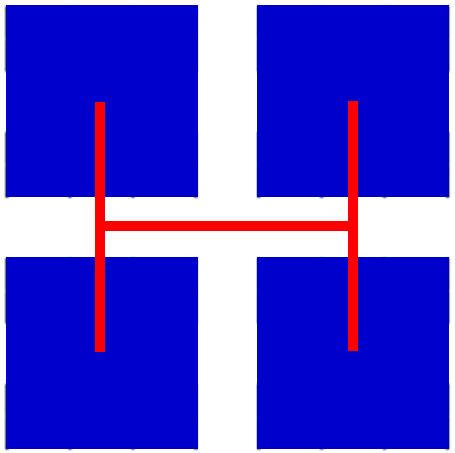




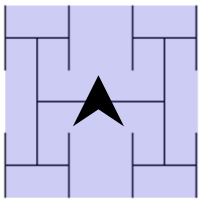


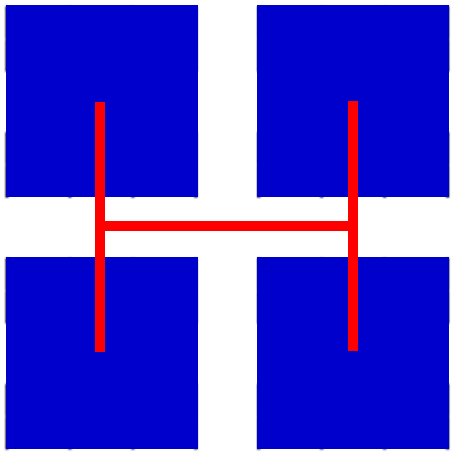
```
draw_shape(tt)
```



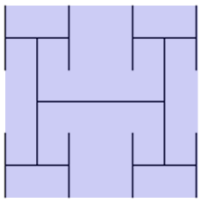


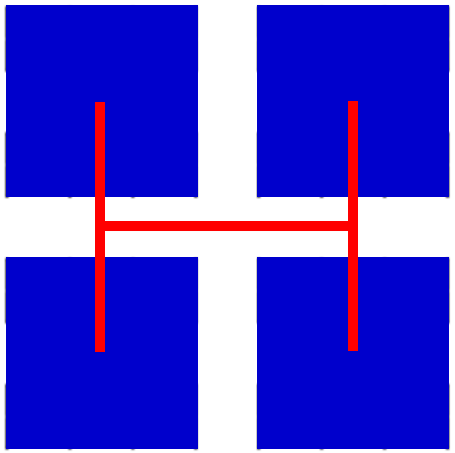
```
def my_function(tt):
```



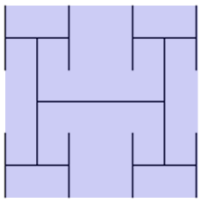


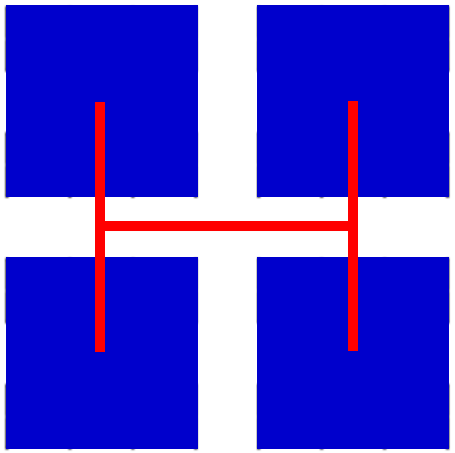
```
def my_function(tt):
    draw_shape(tt)
    tt.backward(100)
    draw_shape(tt)
    tt.forward(50)
    tt.right(90)
    tt.forward(100)
    tt.left(90)
    tt.forward(50)
    draw_shape(tt)
    tt.backward(100)
    draw_shape(tt)
```





```
def draw_shape(tt,N):
    draw_shape(tt,N-1)
    tt.backward(100)
    draw_shape(tt,N-1)
    tt.forward(50)
    tt.right(90)
    tt.forward(100)
    tt.left(90)
    tt.forward(50)
    draw_shape(tt,N-1)
    tt.backward(100)
    draw_shape(tt,N-1)
```

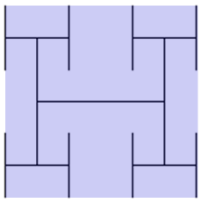


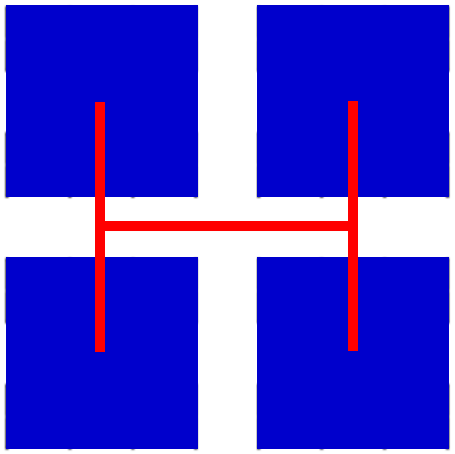


```
def draw_shape(tt,N):
    draw_shape(tt,N-1)
    tt.backward(100)
    draw_shape(tt,N-1)
    tt.forward(50)
    tt.right(90)
    tt.forward(100)
    tt.left(90)
    tt.forward(50)
    draw_shape(tt,N-1)
    tt.backward(100)
    draw_shape(tt,N-1)
```

What is our stopping condition?

N = 1

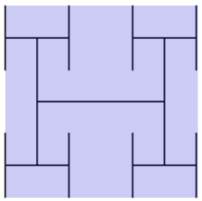


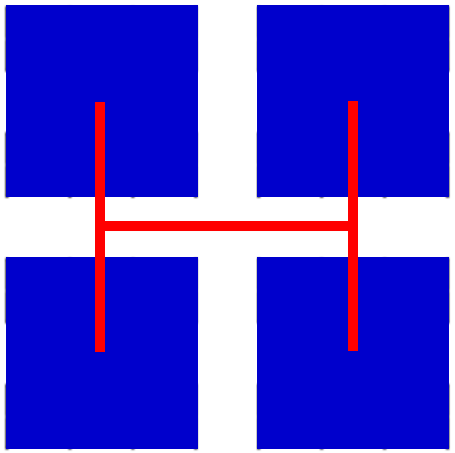


Let's try this …

Hshape01.py

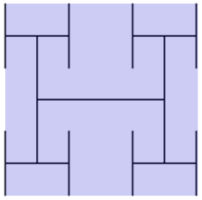
```
def draw_shape(tt,N):
    if N == 1:
        tt.backward(100)
        tt.forward(50)
        tt.right(90)
        tt.forward(100)
        tt.left(90)
        tt.forward(50)
        tt.backward(100)
    else:
        draw_shape(tt,N-1)
        tt.backward(100)
        draw_shape(tt,N-1)
        tt.forward(50)
        tt.right(90)
        tt.forward(100)
        tt.left(90)
        tt.forward(50)
        draw_shape(tt,N-1)
        tt.backward(100)
        draw_shape(tt,N-1)
```

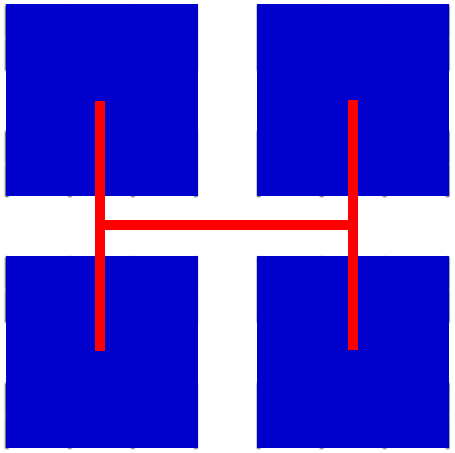




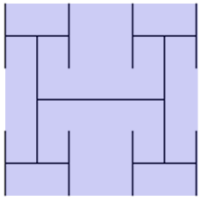
What about the size?

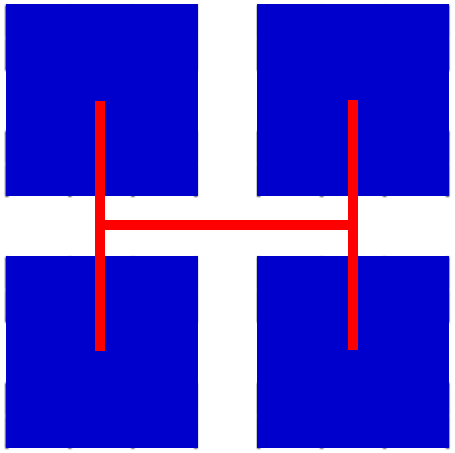
```
def draw_shape(tt,N):
    if N == 1:
        tt.backward(100)
        tt.forward(50)
        tt.right(90)
        tt.forward(100)
        tt.left(90)
        tt.forward(50)
        tt.backward(100)
    else:
        draw_shape(tt,N-1)
        tt.backward(100)
        draw_shape(tt,N-1)
        tt.forward(50)
        tt.right(90)
        tt.forward(100)
        tt.left(90)
        tt.forward(50)
        draw_shape(tt,N-1)
        tt.backward(100)
        draw_shape(tt,N-1)
```





```
def draw_shape(tt,N,size):
    if N == 1:
        tt.backward(size)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        tt.backward(size)
    else:
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
```





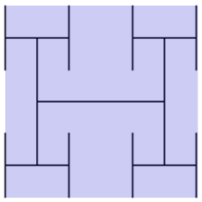
Let's try this …

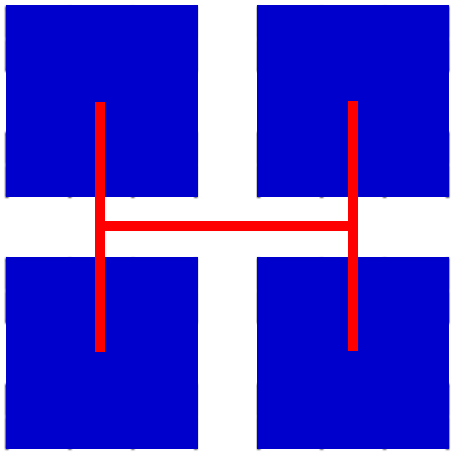
Hshape02.py

```
def draw_shape(tt,N,size):
    if N == 1:
        tt.backward(size)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        tt.backward(size)
    else:
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
```

What was my assumption of what `draw_shape(tt,N-1,size/2)` does?

```
# This function draws the
# shape below and …
```



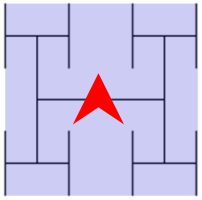


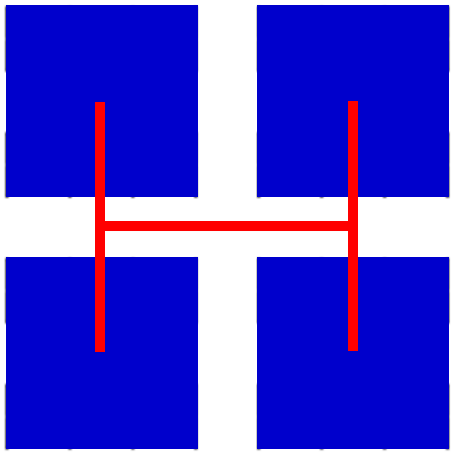
What is the function specification?

```
def draw_shape(tt,N,size):
    else:
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
```

What was my assumption of what `draw_shape(tt,N-1,size/2)` does?

```
# This function draws the
# shape below and the turtle
# starts and ends in middle,
# facing N
```

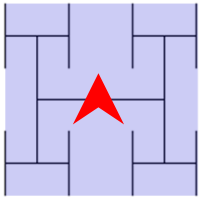


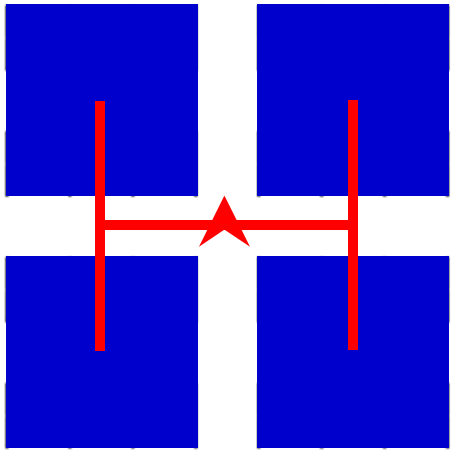


How do we need to modify our code?

```
def draw_shape(tt,N,size):
    else:
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
```

This function draws the
shape below and the turtle
starts and ends in middle,
facing N



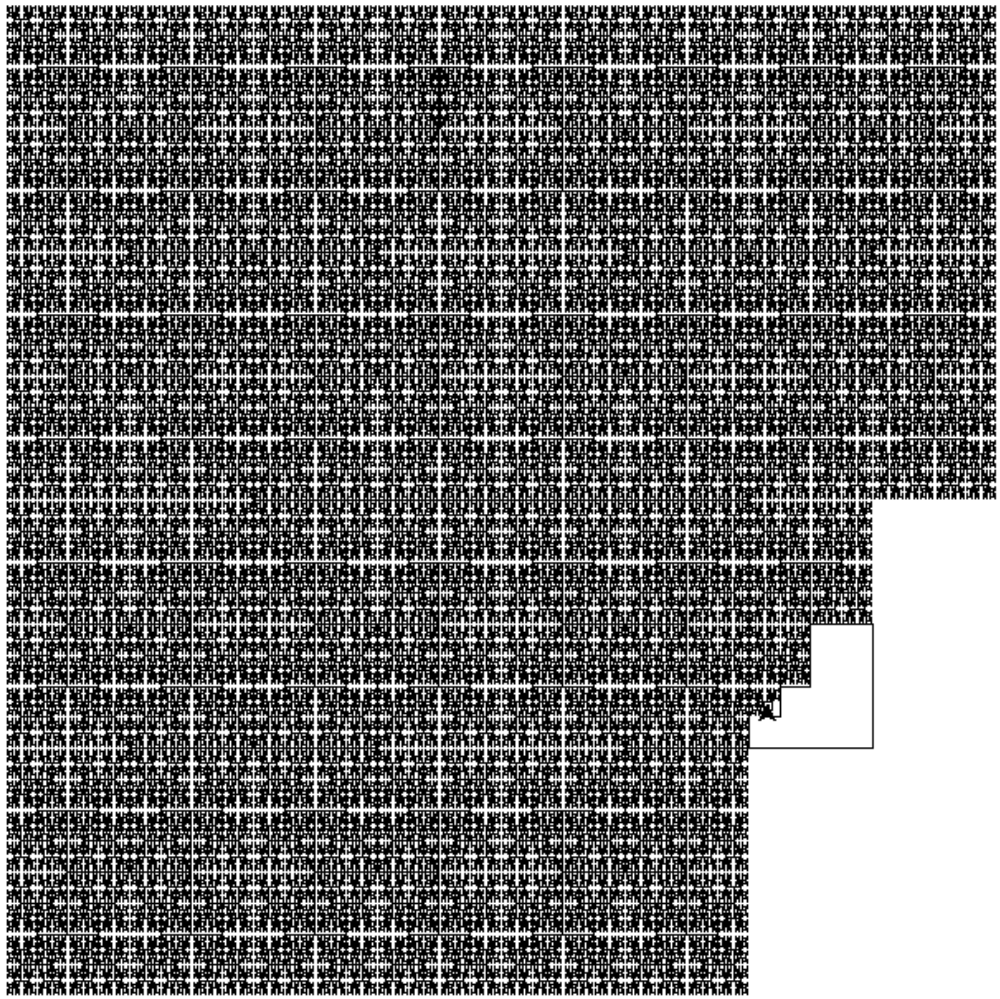


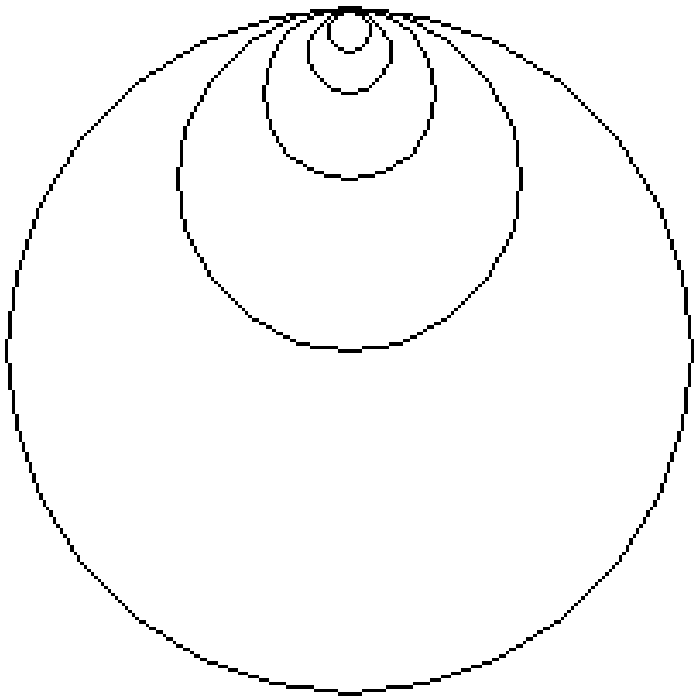
```
def draw_shape(tt,N,size):
    else:
        tt.left(90)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.left(90)
        tt.forward(size/2)
        tt.right(90
```

Let's try this …

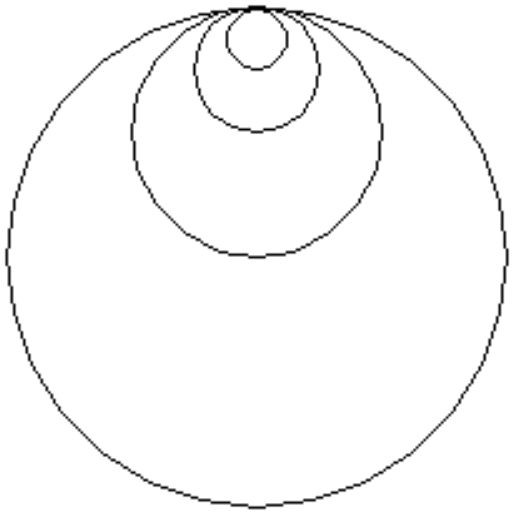
Hshape04.py

```
def draw_shape(tt,N):
    if N == 1:
        tt.left(90)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size/2)
        tt.backward(100)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        tt.backward(size)
        tt.forward(size/2)
        tt.left(90)
        tt.forward(size/2)
        tt.right(90)
    else:
        tt.left(90)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.right(90)
        tt.forward(size)
        tt.left(90)
        tt.forward(size/2)
        draw_shape(tt,N-1,size/2)
        tt.backward(size)
        draw_shape(tt,N-1,size/2)
        tt.forward(size/2)
        tt.left(90)
        tt.forward(size/2)
        tt.right(90
```

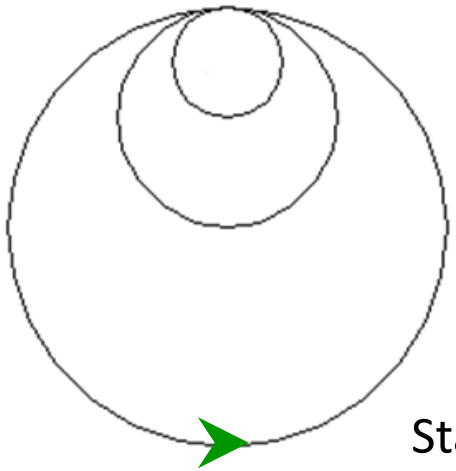




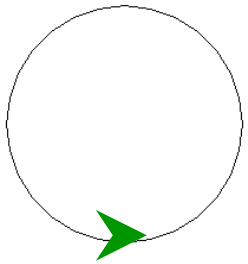
Draw this recursive shape ….



draw4Circles(t,radius)



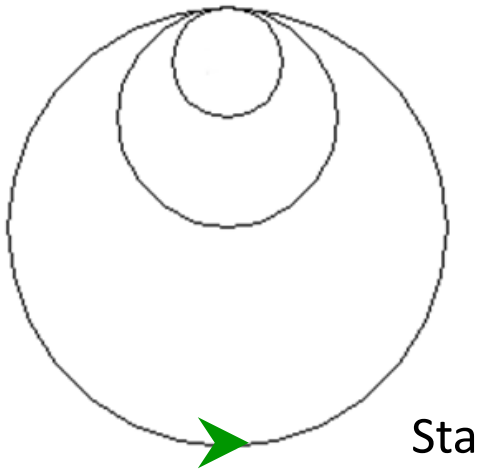
draw3Circles(t,radius)



t.circle(radius)

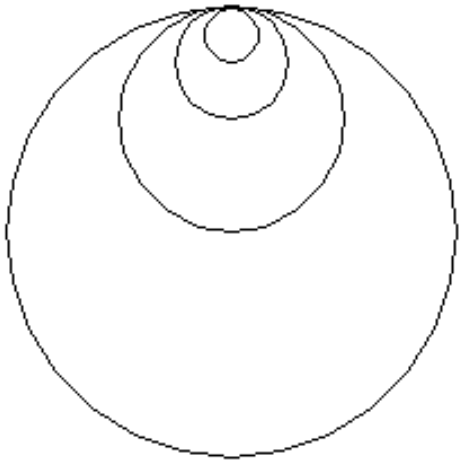
circles01.py

```
def draw4Circles(t,radius):
```

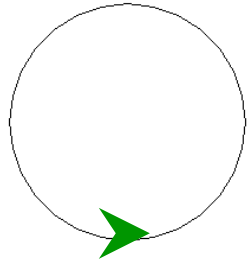


Start/end

draw3Circles(t,radius)



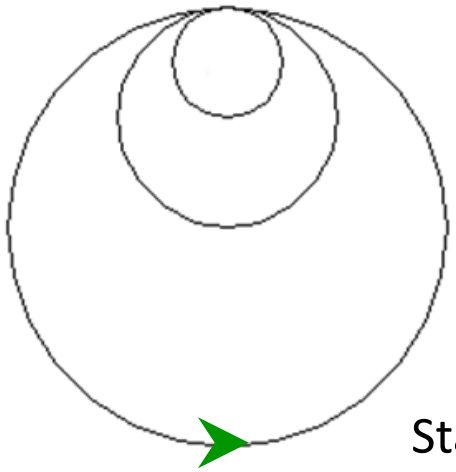
How can we implement this function?



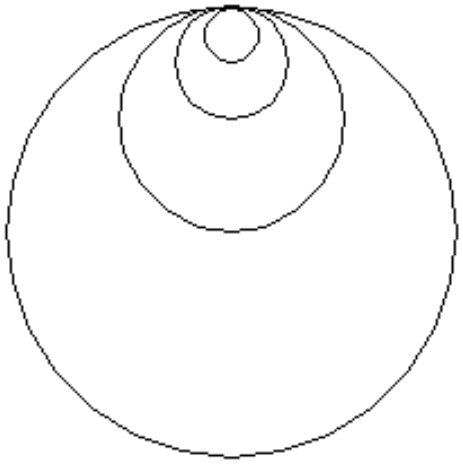
Start/end

t.circle(radius)

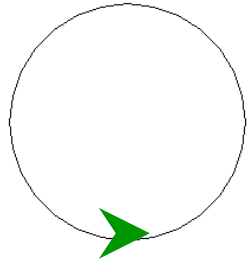
```
def draw4Circles(t,radius):
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    draw3Circles(t,radius//2)
```



draw3Circles(t,radius)

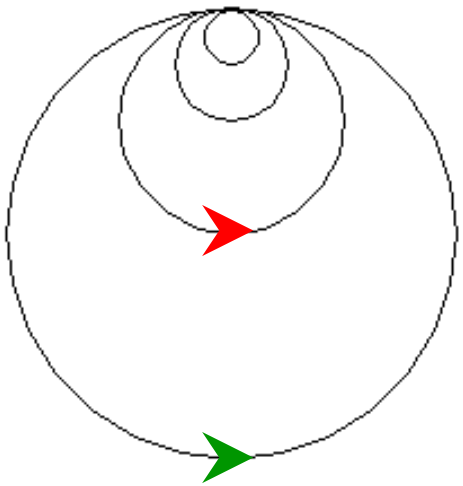


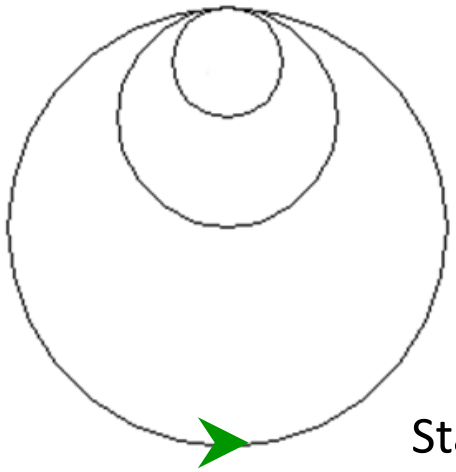
Can we make this recursive now?



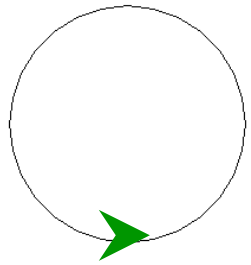
t.circle(radius)

```
def draw4Circles(t,radius):
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    draw3Circles(t,radius//2)
```





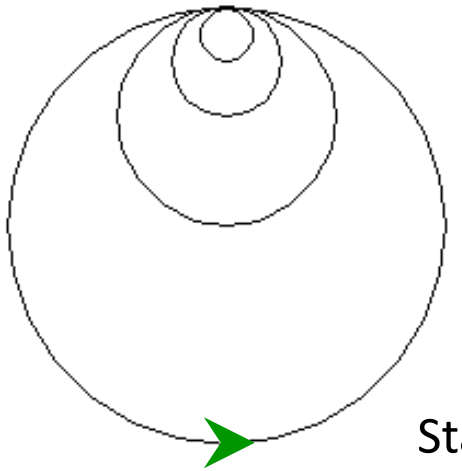
draw3Circles(t,radius)



t.circle(radius)

```
def draw4Circles(t,radius):
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    draw3Circles(t,radius//2)
    t.penup()
    t.right(90)
    t.forward(radius)
    t.left(90)
    t.pendown
```

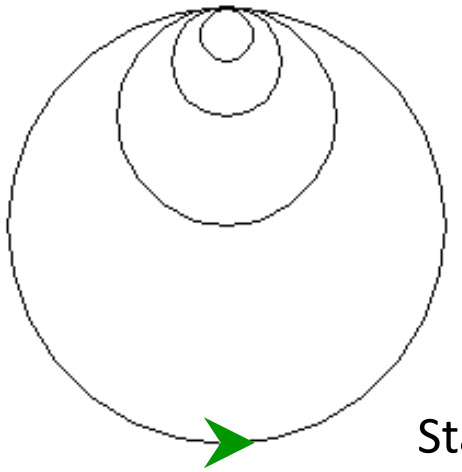
circles02.py



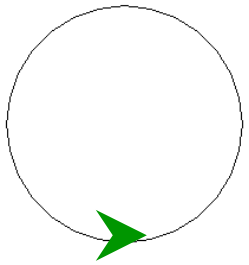
draw4Circles(t,radius)

```
def draw5Circles(t,radius):
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    draw4Circles(t,radius//2)
    t.penup()
    t.right(90)
    t.forward(radius)
    t.left(90)
    t.pendown
```

circles03.py



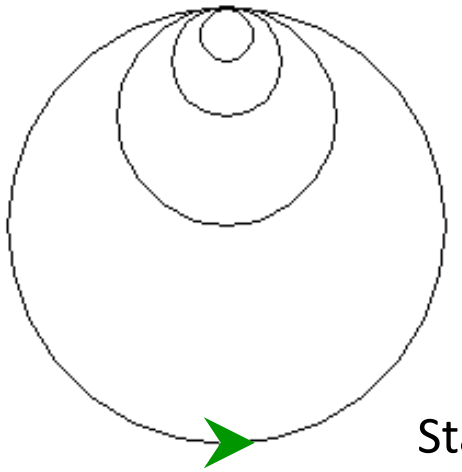
draw4Circles(t,radius)



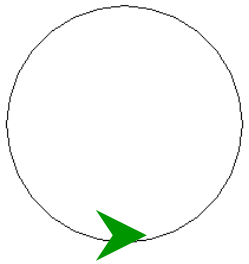
t.circle(radius)

```
def draw5Circles(t,radius):
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    draw4Circles(t,radius//2)
    t.penup()
    t.right(90)
    t.forward(radius)
    t.left(90)
    t.pendown
```

How do we make this recursive?



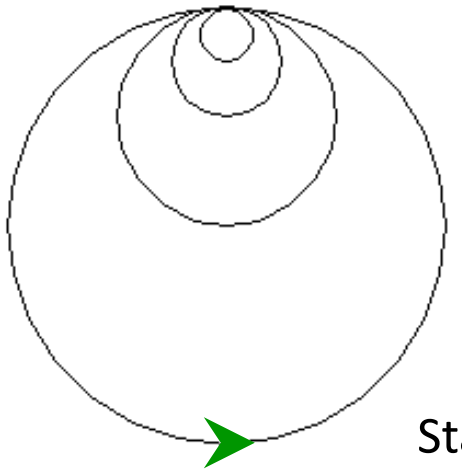
draw4Circles(t,radius)



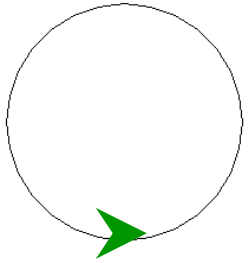
t.circle(radius)

```
def drawNCircles(t,radius,N):
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    drawNCircles(t,radius//2,N-1)
    t.penup()
    t.right(90)
    t.forward(radius)
    t.left(90)
    t.pendown
```

What is the stopping condition?



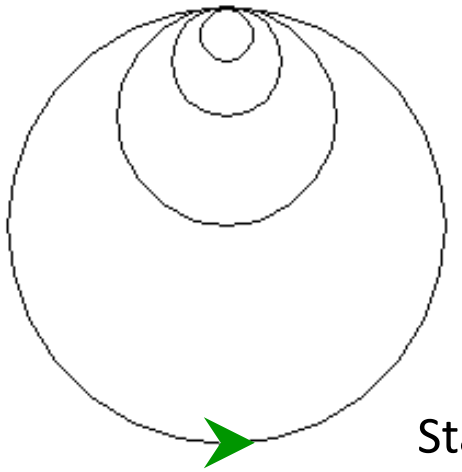
drawNCircles(t,radius,4)



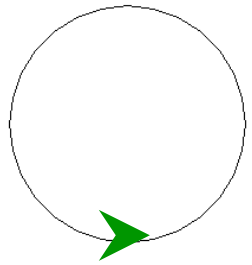
t.circle(radius)

```
def drawNCircles(t,radius,N):
  if N == 0:
    pass
  else:
    t.circle(radius)
    t.penup()
    t.left(90)
    t.forward(radius)
    t.right(90)
    t.pendown()
    drawNCircles(t,radius//2,N-1)
    t.penup()
    t.right(90)
    t.forward(radius)
    t.left(90)
    t.pendown
```

circles04.py

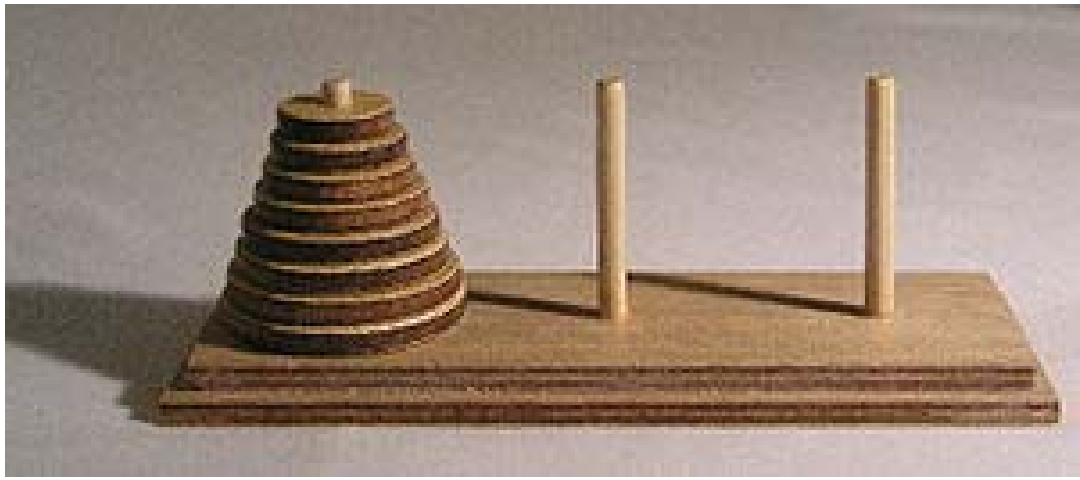


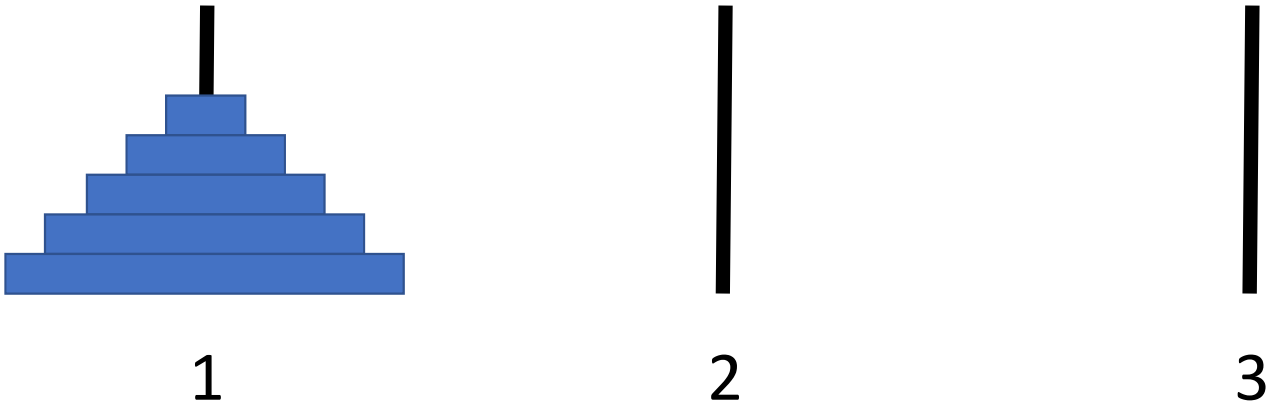
draw4Circles(t,radius)



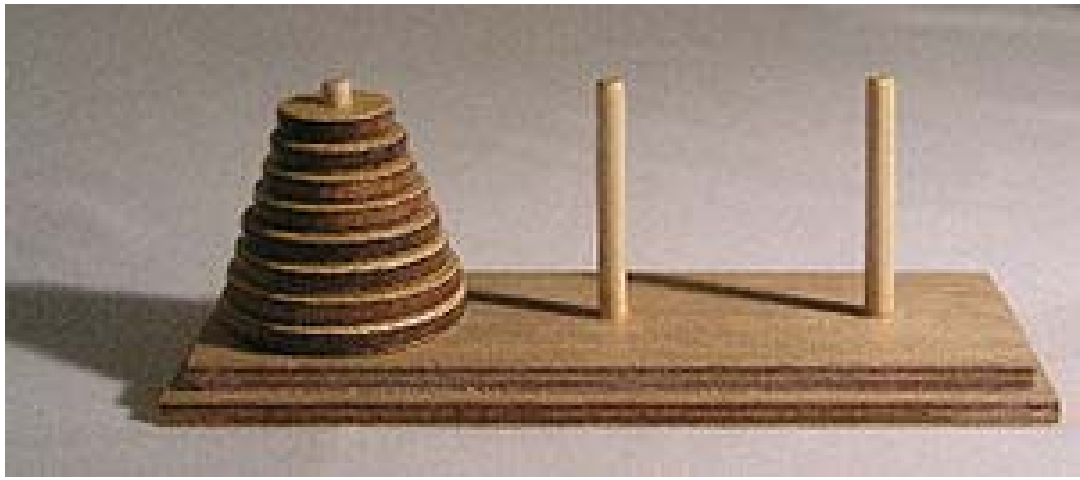
t.circle(radius)

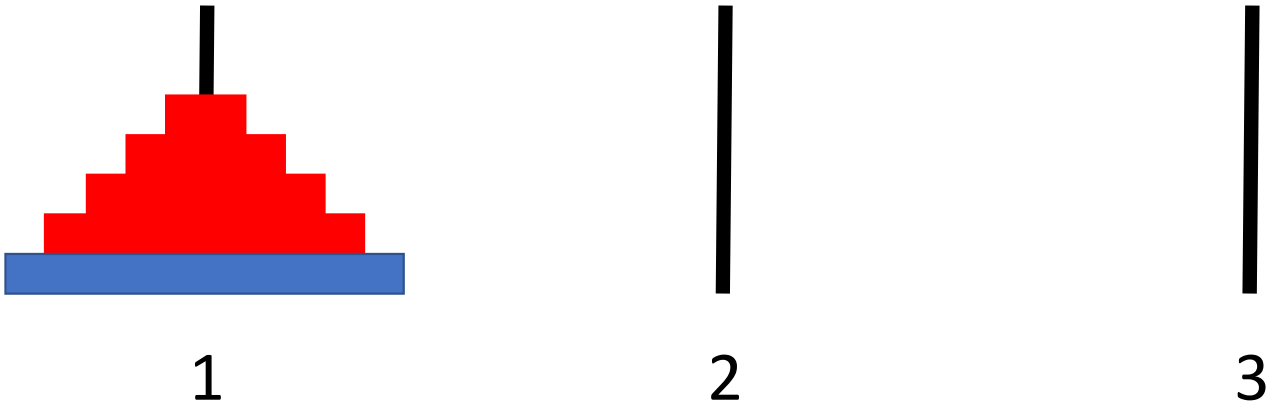
Tower of Hanoi



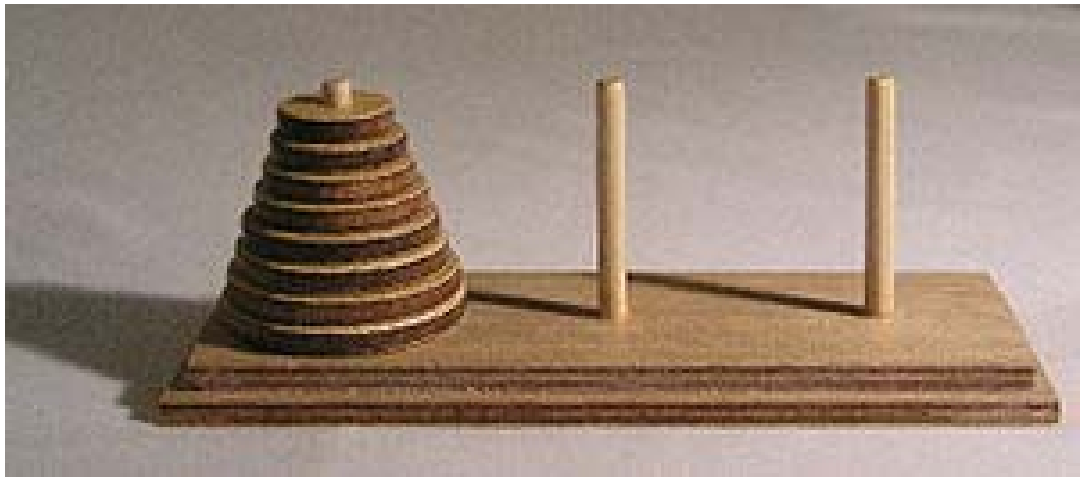


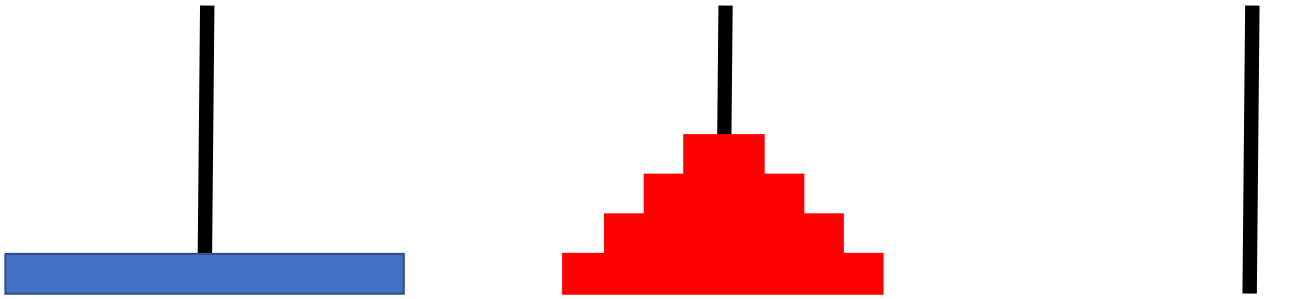
Tower of Hanoi



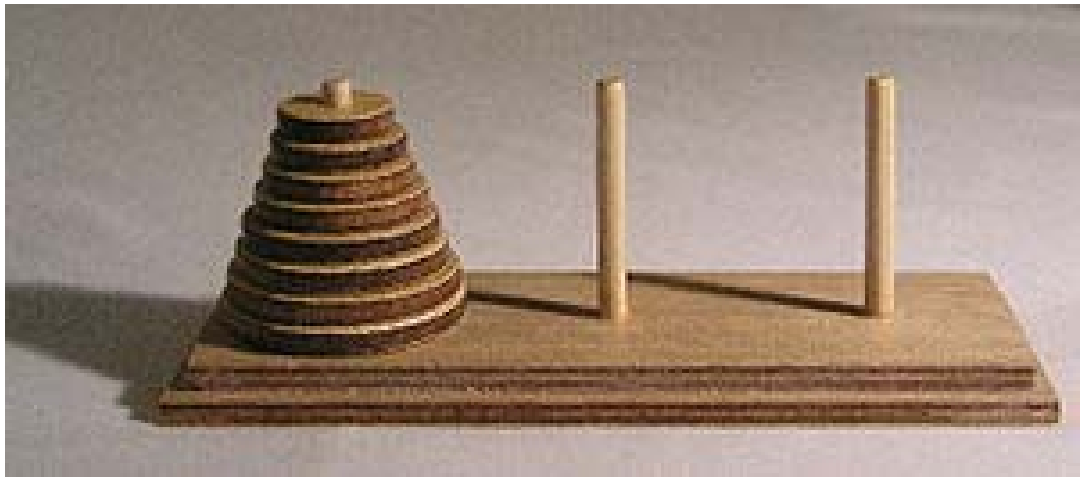


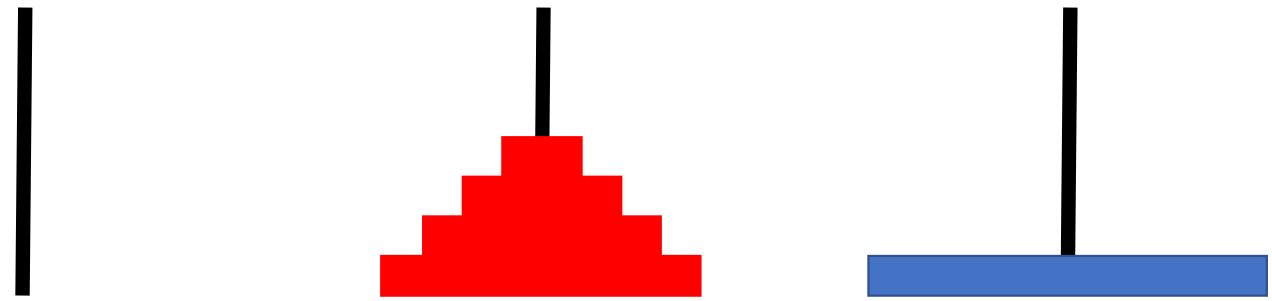
Tower of Hanoi



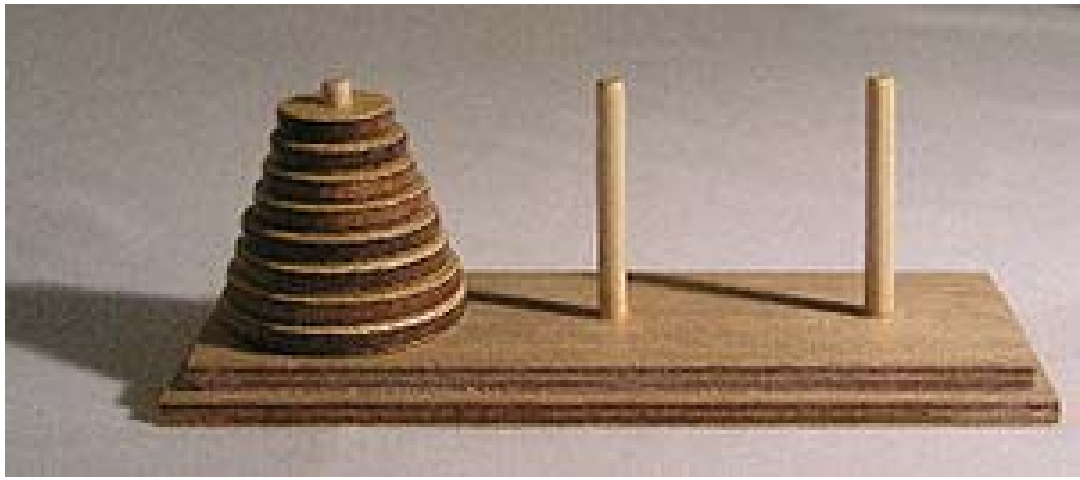


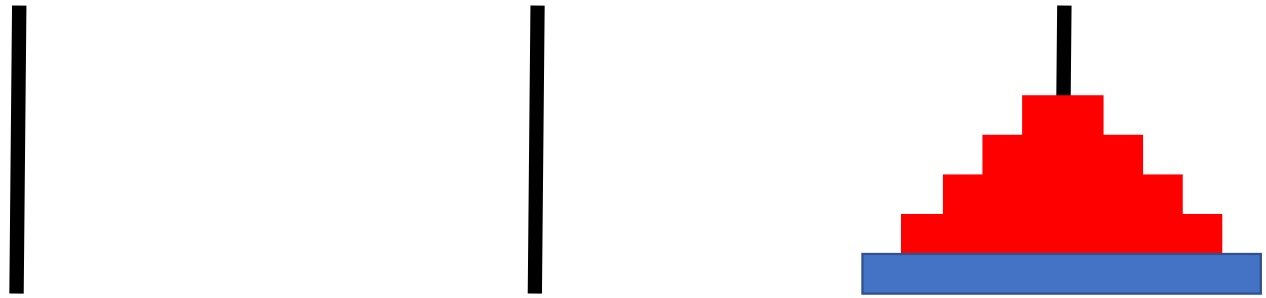
Tower of Hanoi





Tower of Hanoi

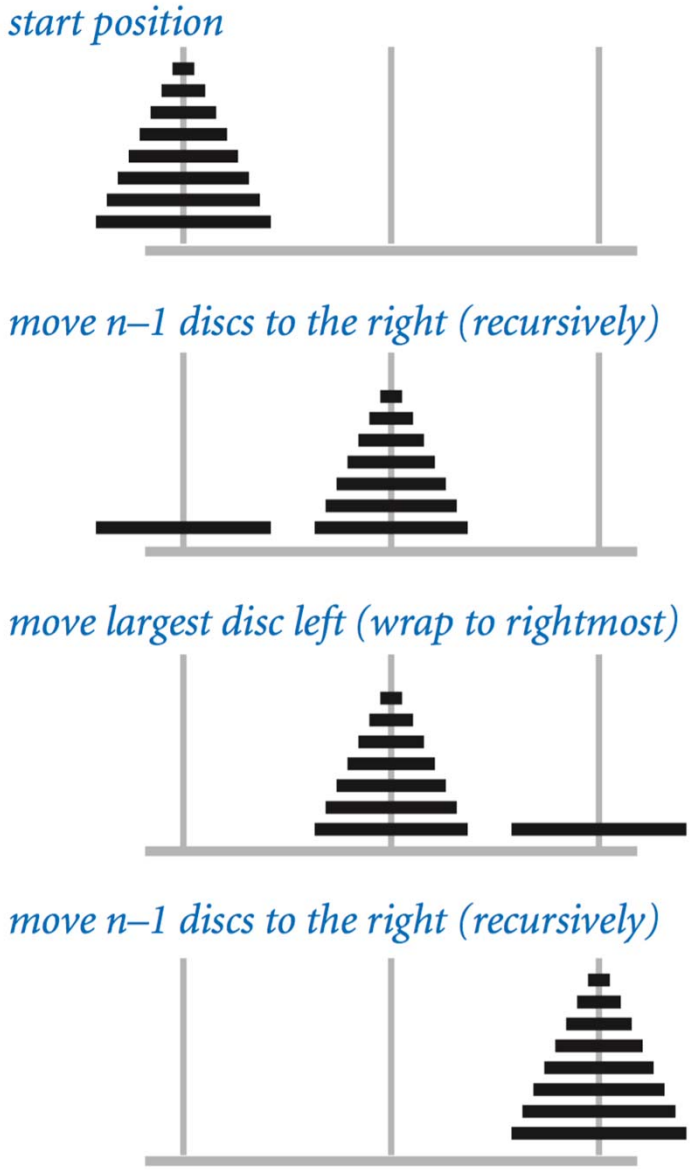




Tower of Hanoi

```
def move_tower_5(1,3):
      move_tower_4(1,2)
      move_bottom(1,3)
      move_tower_4(2,3)
```

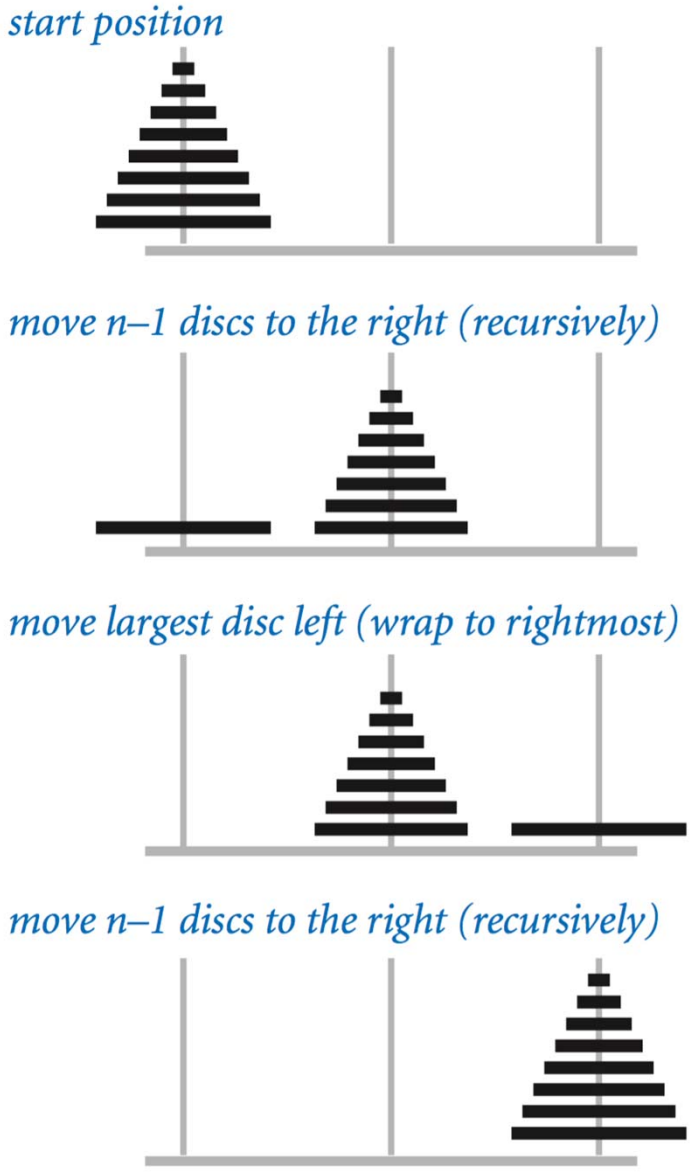
```
def move_tower_4(start,end):
      …
```



Tower of Hanoi

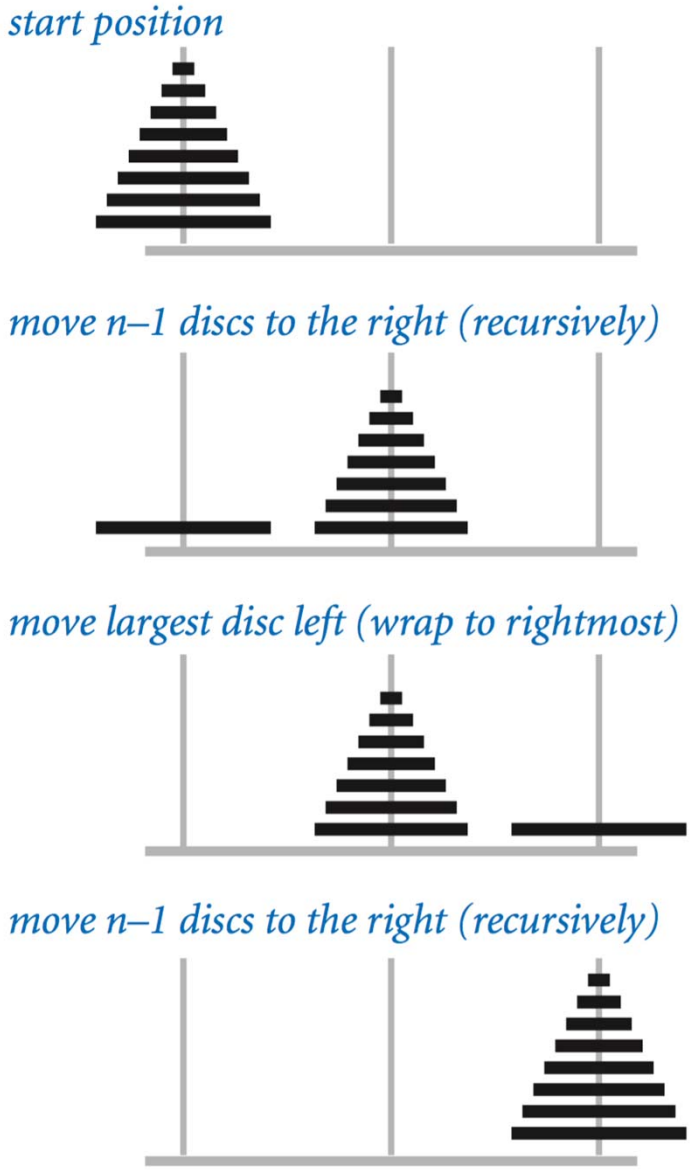
```
def move_tower_5(start,end):
      move_tower_4(start,xtra)
      move_bottom(start,end)
      move_tower_4(xtra,end)
```

```
def move_tower_4(start,end):
      …
```



Tower of Hanoi

```
def move_tower(N,start,end):
      move_tower(N-1,start,xtra)
      move_bottom(start,end)
      move_tower(N-1,xtra,end)
```



Tower of Hanoi

```
def move_tower(N,start,end):
    if N == 1:
      move_bottom(start,end)
    else:
      move_tower(N-1,start,xtra)
      move_bottom(start,end)
      move_tower(N-1,xtra,end)
```

towerhanoi1.py