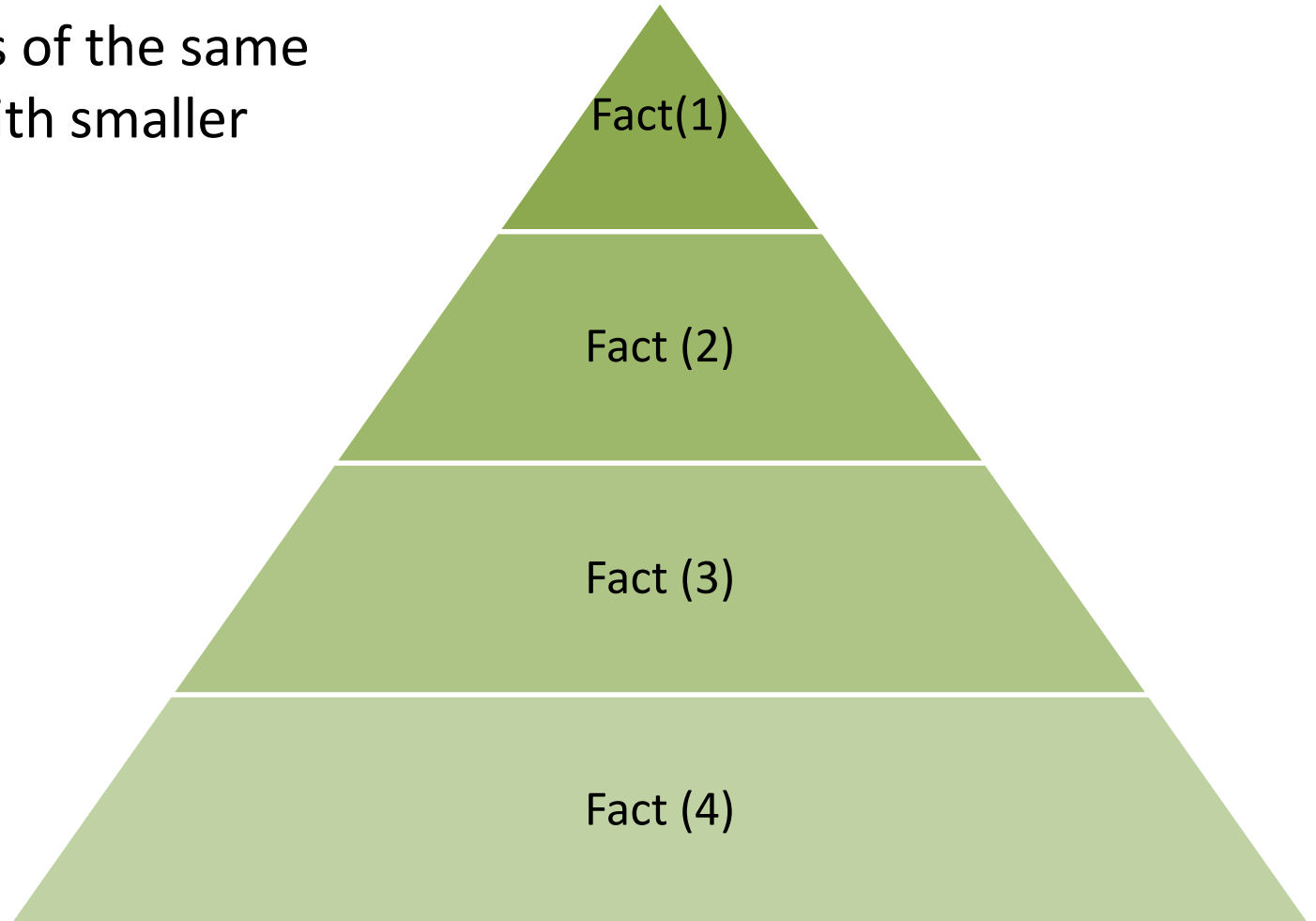

Recursive graphics

Lecture 06.02

By Marina Barsky

Recall: recursion

Pile of copies of the same function – with smaller inputs



Using turtle canvas as sketchpad

```
from turtle import *
```

```
reset()
```

```
forward(50)
```

```
left(90)
```

```
forward(100)
```

```
right(90)
```

```
color('blue')
```

```
width(5)
```

```
forward(100)
```

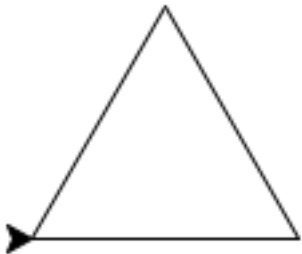
```
circle(50)
```

```
done()
```

Identifying repeating patterns

- Drawing a triangle

```
def tri():  
    """ draws a  
    triangle  
    """  
    forward(100)  
    left(120)  
    forward(100)  
    left(120)  
    forward(100)  
    left(120)
```



Identifying repeating patterns

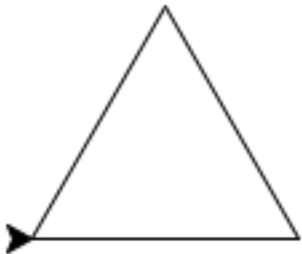
- Drawing a triangle ... recursively

```
def tri():  
    """ draws a  
    triangle  
    """  
    forward(100)  
    left(120)  
    forward(100)  
    left(120)  
    forward(100)  
    left(120)
```



```
def tri_rec(n):  
    if n == 0:  
        return  
    else:  
        forward(100)  
        left(120)  
        tri_rec(n-1)
```

tri_rec(3)



Could we create *any* regular n-gon?

Identifying repeating patterns

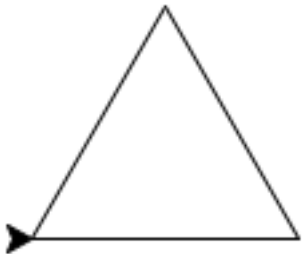
- Drawing a triangle ... recursively

```
def tri():  
    """ draws a  
    triangle  
    """  
    forward(100)  
    left(120)  
    forward(100)  
    left(120)  
    forward(100)  
    left(120)
```



```
def tri_rec(n):  
    if n == 0:  
        return  
    else:  
        forward(100)  
        left(120)  
        tri_rec(n-1)
```

tri_rec(3)



Generic n-gon

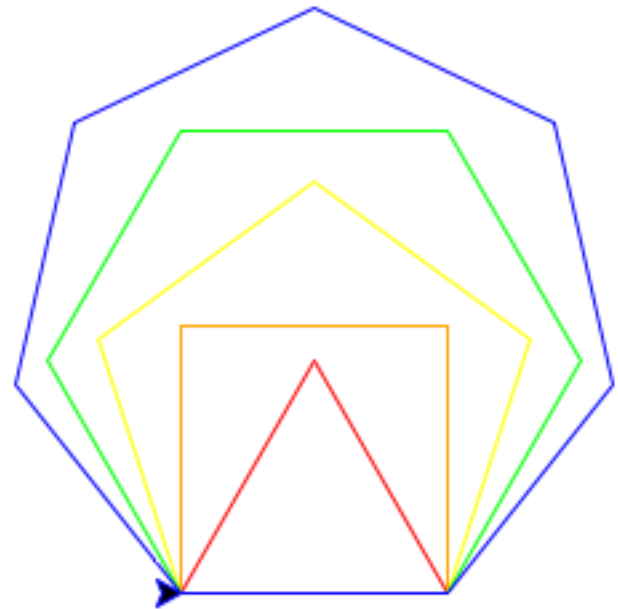
A default parameter value!

```
def ngon(n, cur_side=0):  
    """ A simple recursive function to create an arbitrary  
        n-sided polygon  
        Parameters:  
        n - number of sides of the polygon  
        cur_side - currently drawing side used by recursion  
    """  
    if cur_side >= n:  
        return  
    else:  
        forward(100)  
        left(360/n)  
        ngon(n, cur_side+1)
```

How many degrees should we turn?

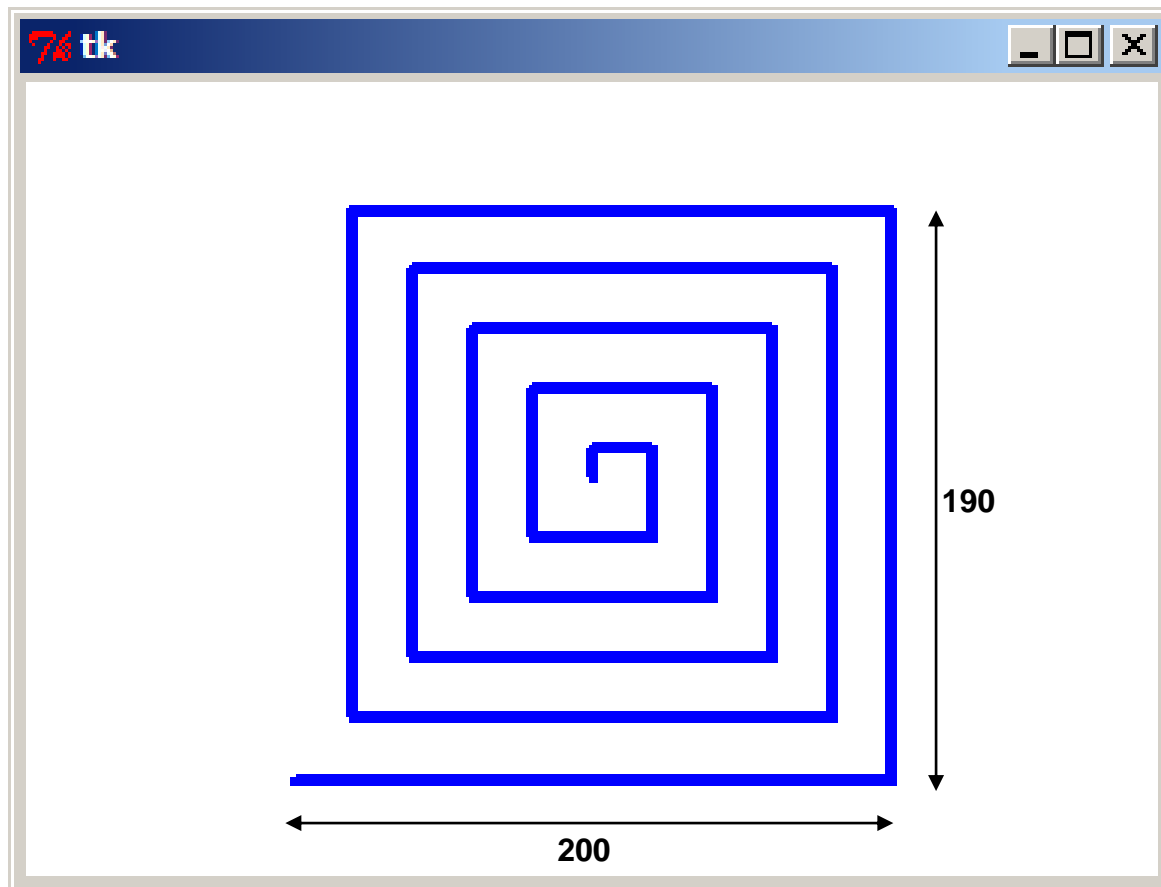
ngon(n)

```
pencolor ( ' red' )  
ngon (3)  
pencolor ( ' orange' )  
ngon (4)  
pencolor ( ' yellow' )  
ngon (5)  
pencolor ( ' green' )  
ngon (6)  
pencolor ( ' blue' )  
ngon (7)
```

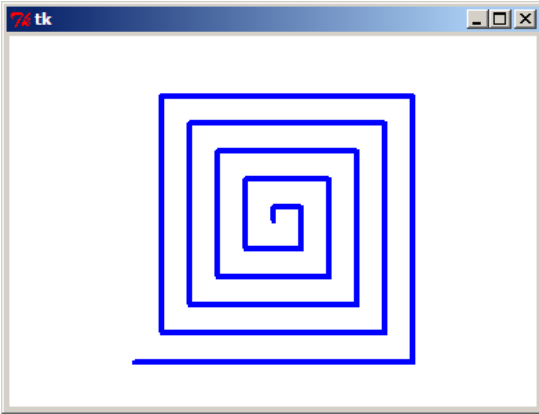


Drawing Spirals

Any self-similarity here?



Designing recursive drawing



Base Case:

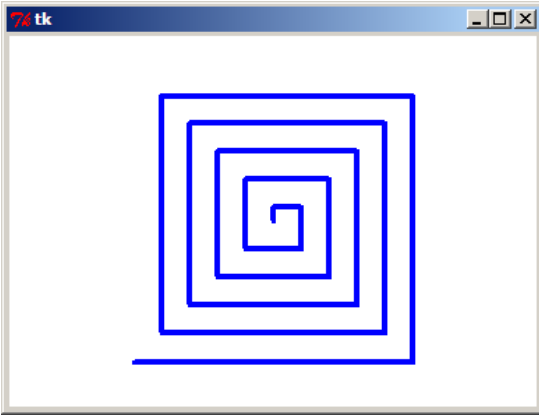
Recursive Step:

Designing any recursive program boils down to the same two pieces

Think about the **SIMPLEST POSSIBLE** case!

Do **ONLY ONE STEP**, and let recursion do the rest...

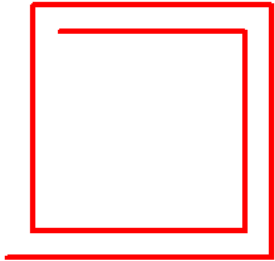
Recursion: stop when too small



Designing any recursive program boils down to the same two pieces

```
def spiral_square(side_len):  
Base Case: → if side_len < 1:  
               return  
               else:  
                 forward(x)  
                 left(90)  
Recursive Step: → spiral_square(side_len - 10)
```

Recursion: stop after n steps



Designing any recursive program boils down to the same two pieces

`spiral_square(200, 7)`

```
def spiral_square(side_len, n):  
Base Case: → if n == 0:  
               return  
               else:  
                 forward(x)  
                 left(90)  
Recursive Step: → spiral_square(side_len-10, n-1)
```

Exercise in 2D

What does function *tea* draw?

```
def tea(size):  
    """ Mystery! """  
    forward(size)  
  
    left(90)  
    forward(size/2)  
    backward(size/2)  
    right(90)  
  
    right(90)  
    forward(size/2)  
    backward(size/2)  
    left(90)  
  
    backward(size)
```

Exercise in 2D

What if I want to draw a double-headed axe?

```
def tea(size):  
    """ Letter T """  
    forward(size)  
  
    left(90)  
    forward(size/2)  
    backward(size/2)  
    right(90)  
  
    right(90)  
    forward(size/2)  
    backward(size/2)  
    left(90)  
  
    backward(size)
```



An ornated golden Minoan double axe, often spuriously called a labrys.

Extend T



Simpler tea

```
def tea(size):  
    """ letter T """  
    forward(size)  
  
    left(90)  
    forward(size/2)  
    backward(size/2)  
    small_tea(size/2)  
    right(90)  
  
    right(90)  
    forward(size/2)  
    backward(size/2)  
    small_tea(size/2)  
    left(90)  
  
    backward(size)
```

```
def small_tea(size):  
    forward(size)  
  
    left(90)  
    forward(size / 2)  
    backward(size / 2)  
    right(90)  
  
    right(90)  
    forward(size / 2)  
    backward(size / 2)  
    left(90)  
  
    backward(size)
```

But 2 functions are the same –
recursion!

Recursive tea

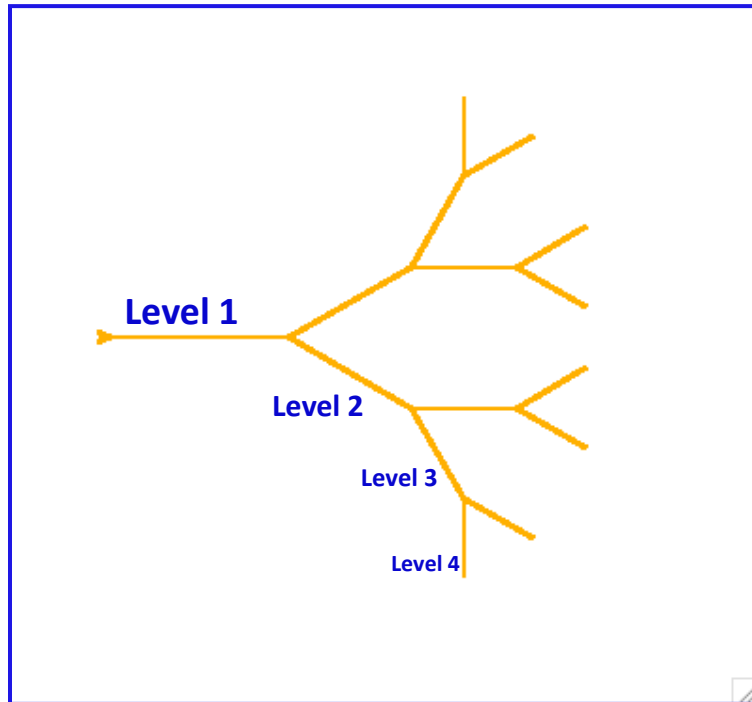
```
def tea(size, iteration):  
    """ letter T """  
    if iteration == 0:  When to stop  
        return  
    forward(size)  
  
    left(90)  
    forward(size / 2)  
    backward(size / 2)  
    tea(size / 2, iteration - 1)  When finished  
    right(90)   
    right(90)  
    forward(size / 2)  
    backward(size / 2)  
    tea(size / 2, iteration - 1)  
    left(90)  
  
    backward(size)
```

When finished
one branch –
make it into a
smaller T!

Fractals: recursive drawings

- When you look at fractal it has the same basic shape no matter how much you magnify it
- Nature:
 - Coastlines of continents
 - Snowflakes
 - Mountains
 - Trees or shrubs
- The fractal nature of these natural phenomena makes it possible for programmers to generate very realistic looking scenery for computer-generated movies.

Fractal: side-view tree

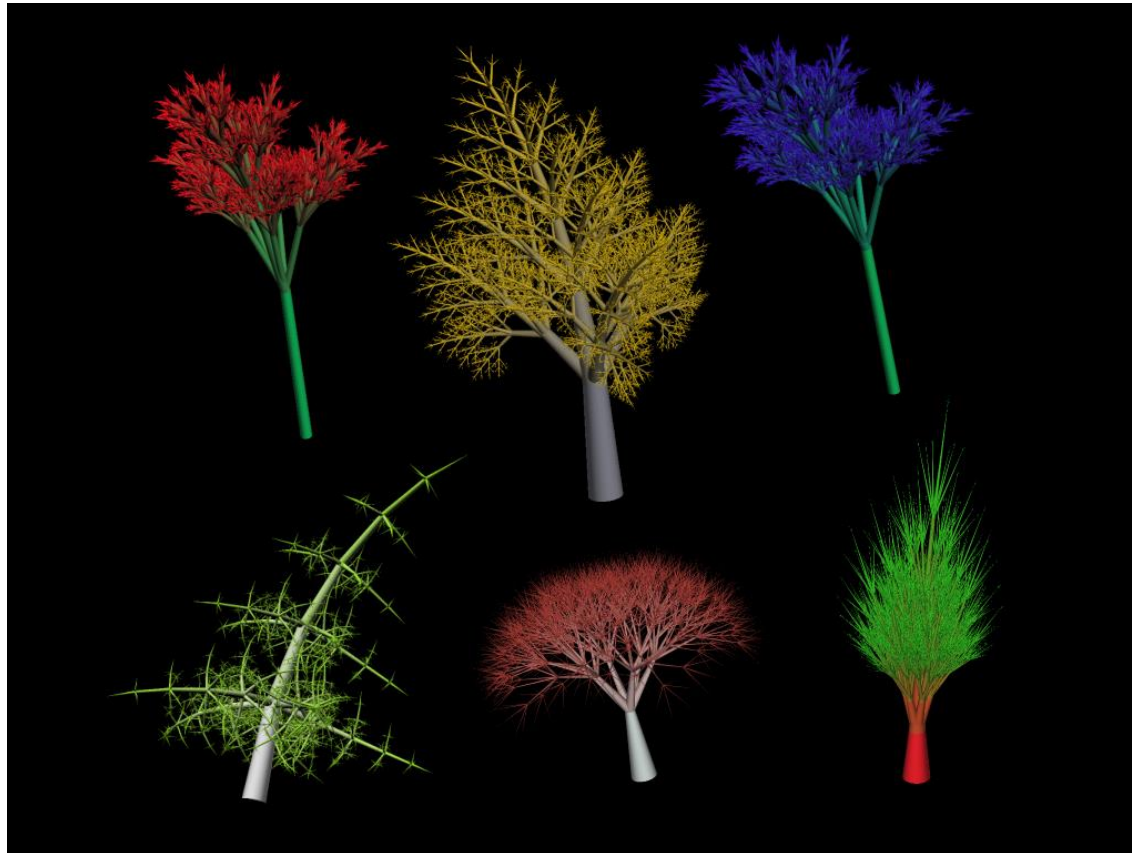


`sv_tree(100, 4)`

- How to describe a tree using a fractal vocabulary?
- A tree is a trunk with:
 - Smaller tree to the left
 - Smaller tree to the right
- We can apply the recursive definition of a tree to both the smaller left and right trees.

`sv_tree(trunk_size, levels)`

Recursion in nature



Key: self-similarity

Fractals: emerging patterns

Mandelbrot set:

- <https://www.youtube.com/watch?v=2JUAojvFpCo>

Minskytron:

- <https://www.youtube.com/watch?v=IXsVWwPa7bc>
- <https://www.masswerk.at/minskytron/>

The pair of equations can be expressed succinctly as
 $y -= x \gg 4;$
 $x += y \gg 4;$

Demo:

- <http://hope.simons-rock.edu/~mbarsky/intro18/mandel/>
- <http://hope.simons-rock.edu/~mbarsky/intro18/minski/>