

Recursive Functions

Section 13.10 in textbook

Goals

- Introduce the idea of *recursion*
- Explore the idea of recursion in the real world via *fractal images*
- Outline the general structure of a *recursive function*
- Draw *our own fractals* using recursive drawing functions
- Explore more traditional applications of recursion
- Argue for using recursion in solving problems
- Wednesday – Additional examples, recursive data structures, relationship to loops

Functions

Functions:

Are a labeled block of instructions

Instructions in the block can be any instructions allowed by the language

Can take inputs (parameters)

Can return values

Once defined, become a new instruction in the language we can make use of

Recursion

Functions: Labeled block of arbitrary instructions

+

Functions: Once defined, become a new instruction in the language we can make use of

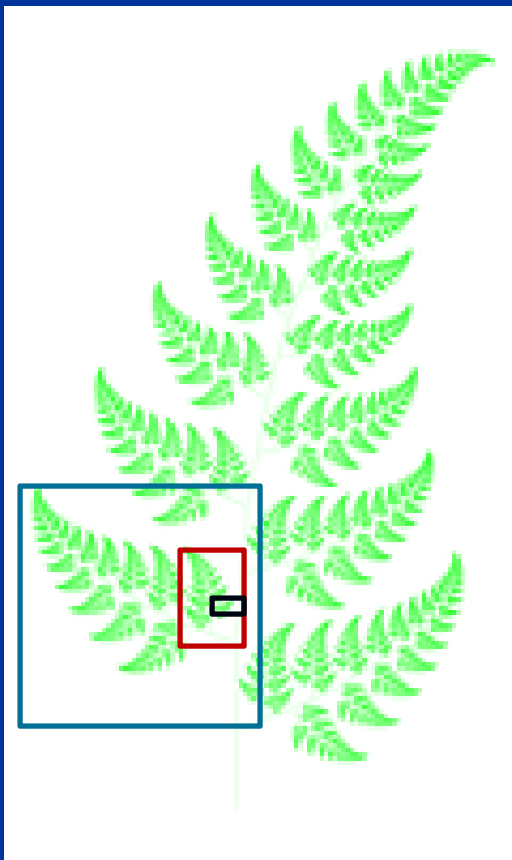
A *recursive function* is when a function uses itself as one of the instructions within its labeled block of instructions (passing in different inputs)

Fractals

- Next few slides:
 - Introduce fractals as a means of visualizing the idea of *recursion* (a process making use of itself)
 - Demonstrate that fractals can be drawn by employing *recursive functions* in the Processing language

Fractal Images – A Way to Visualize Recursion

Fractal Fern Image



A *fractal image* is a visualization of a geometric shape which can be broken apart into pieces, each of which is a reduced size copy of the original (*property of self-similarity*)

For the fractal fern, each component of the fern leaf is actually a smaller copy of the leaf, and each subcomponent is a smaller copy, and so on... until we reach some size limit

Fractals In Nature



Nautilus Shell

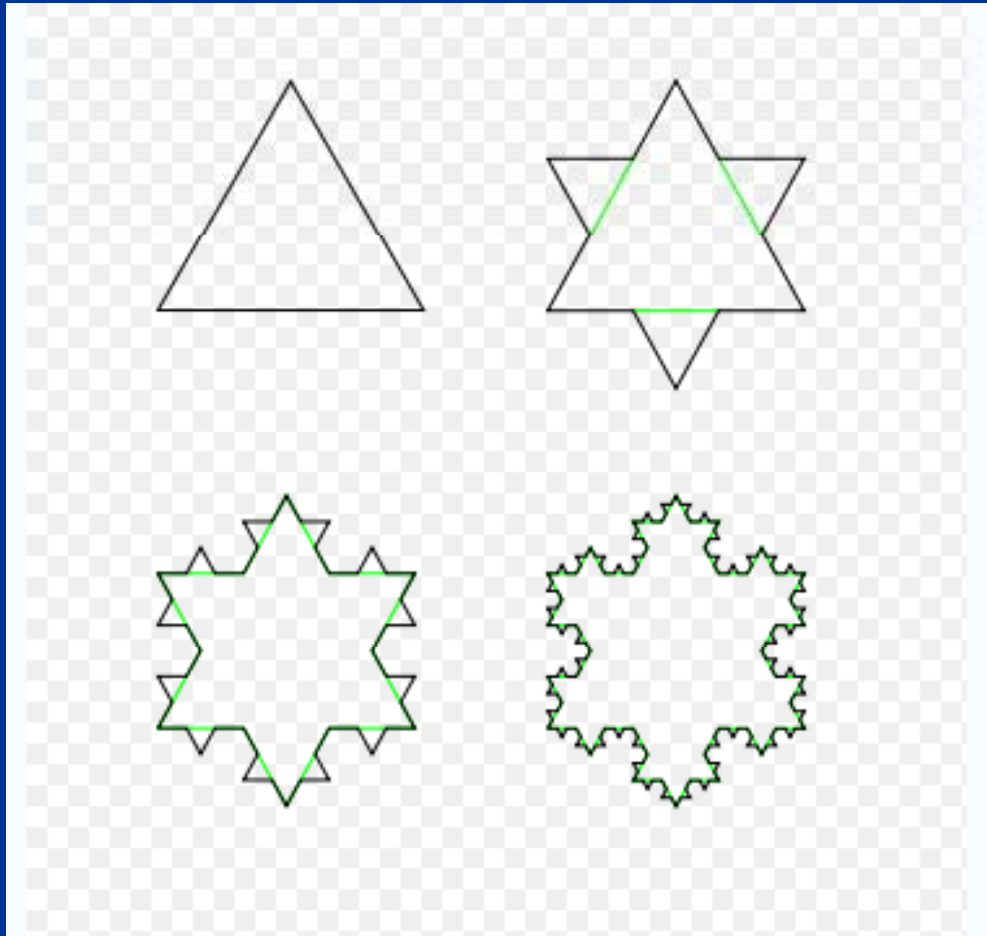


Lightning Strikes



Rivers and Tributaries

Designed Fractals



Koch snowflake

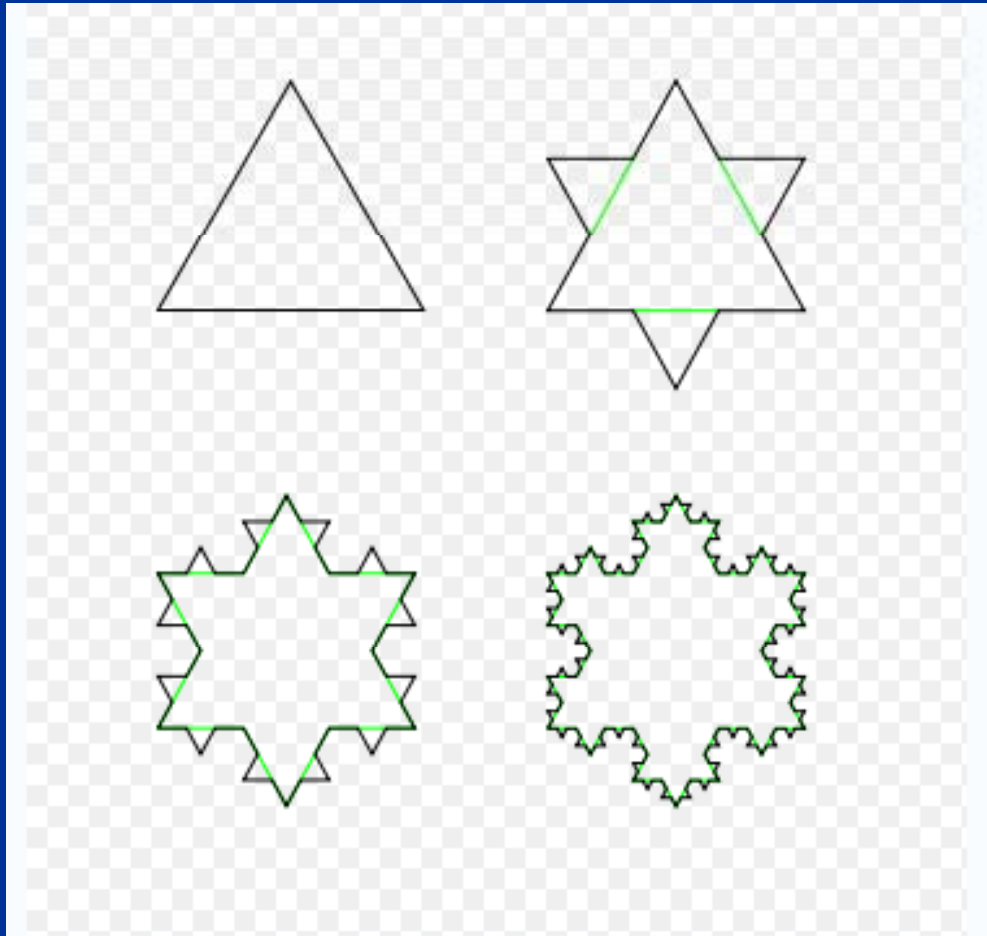
Draw an equilateral triangle

At the center of each side, replace the middle $1/3^{\text{rd}}$ with an equilateral bump

Repeat *ad infinitum* (until it's too small to draw/see!)

Designed Fractals

A conditional for Koch Snowflake



Repeatedly evaluate:

if (we are drawing too small)

{

stop drawing

Base case

}

else

{

-draw component

- request smaller

component to be drawn

Recursive case

}

Recursive Functions

```
return_type name(type input1, type input2, ...) {  
    if (we are at a stopping point) {  
        - do some work  
        - stop  
    }  
    else {  
        - do some work  
        - use function name with different (“smaller”)  
        parameters  
    }  
}
```

Base case

Recursive case

Recursive Functions

An important key idea:

Must eventually make sure the base case test passes!

If not, what do you think happens?

Infinite loop

```
return_type name(type input1, type input2, ...) {  
    if (we are at a stopping point) {  
        - do some work  
        - stop  
    }  
    else {  
        - do some work  
        - use function name with different (“smaller”)  
        parameters  
    }  
}
```

Recursive Drawing Functions

Let's look at some examples with Processing...

- NestedCircles (looks like a bullseye)
- Circles1 (we'll try to figure out what this looks like)
- Circles2 (we'll try to figure out what this looks like)

Recursive Mathematical Functions

Recursive drawing functions allow us to draw fractals – self similar geometric shapes

There are a number of mathematical functions which can be defined recursively:

- computing the factorial of a number
- computing the n th value in the fibonacci sequence
- computing the n th triangular number

Recursive Functions: Math

Example: Factorial

Factorial of a number:

$$1! = 1$$

$$2! = 2$$

$$3! = 6$$

$$4! = 24$$

$$5! = 120$$

Factorial(N) =

Product of 1..N

Recursive Functions: Math

Examples: Factorial

- Factorials have the property that

$$N! = N * (N-1)!$$

$$5! = 120 \quad 4! = 24 \quad \Rightarrow 5! = 5 * 4!$$

- So, if we have a very simple function to compute $(N-1)!$, we could use it to solve $N!$ factorial
- Really solving same problem, but a smaller version

Recursive Functions: Math

Examples: Factorial

Factorial of a number:

$$1! = 1$$

$$2! = 2$$

$$3! = 6$$

$$4! = 24$$

$$5! = 120$$

$$\text{Factorial}(N) =$$

Product of 1..N

$$N! = N * (N-1)!$$

What is base case to test for?

What is base case solution/code?

What is recursive case to test for?

What is recursive case solution/code?

Recursive Functions: Math

Examples: Factorial

```
int factorial(int inputValue)
{
    // base case
    if (inputValue == 1) { return 1; }
    // recursive case
    else { return inputValue * factorial(inputValue -
1); }
}
```

Summary

- *Recursion*: A process to solve a problem which employs itself on a smaller variation of the problem in coming up with a solution
 - *Base case*: An instance of the problem which has a direct solution ; stops the recursion
 - *Recursive case*: An instance of the problem which requires some work + a recursive call
- *Fractal*: Images defined by recursive processes, found in nature!
- Demonstrated *applicability* to a math problems (factorial), can *simplify programming* in some cases