

Recursion

You're asking me for inception. I do hope you understand the *gravity* of that request.

-- Cobb

Common mistakes from last week

- Make sure your if statements are indented properly!

```
def tri(size, fill=False):
    if fill:
        winston.board.fill()
        winston.board.resize(size)
    else:
        winston.board.resize(size)
    winston.board.fill(size)
    if fill:
        winston.board.fill()
```

Common mistakes from last week

- Make sure your if statements are indented properly!

```
def tri(size, fill=False):  
    if fill:  
        winston.begin_fill()  
        winston.forward(size)  
        winston.left(90)  
        winston.forward(size)  
        winston.left(90)  
        winston.forward(size)  
    if fill:  
        winston.end_fill()
```

Comments in code

```
#this is a function that draws a triangle
```

```
def triangle(size)  
    winston.forward(size)  
    winston.left(120)  
    winston.forward(size)  
    winston.left(120)  
    winston.forward(size)  
    winston.left(120)
```

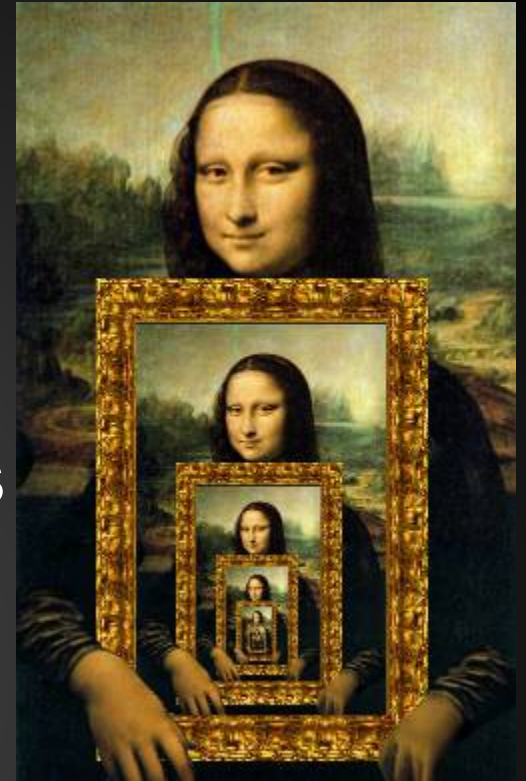
```
triangle(100) #draw a triangle of size 100
```

```
print 'Your triangle is done! #this is not a comment'
```

What is recursion?

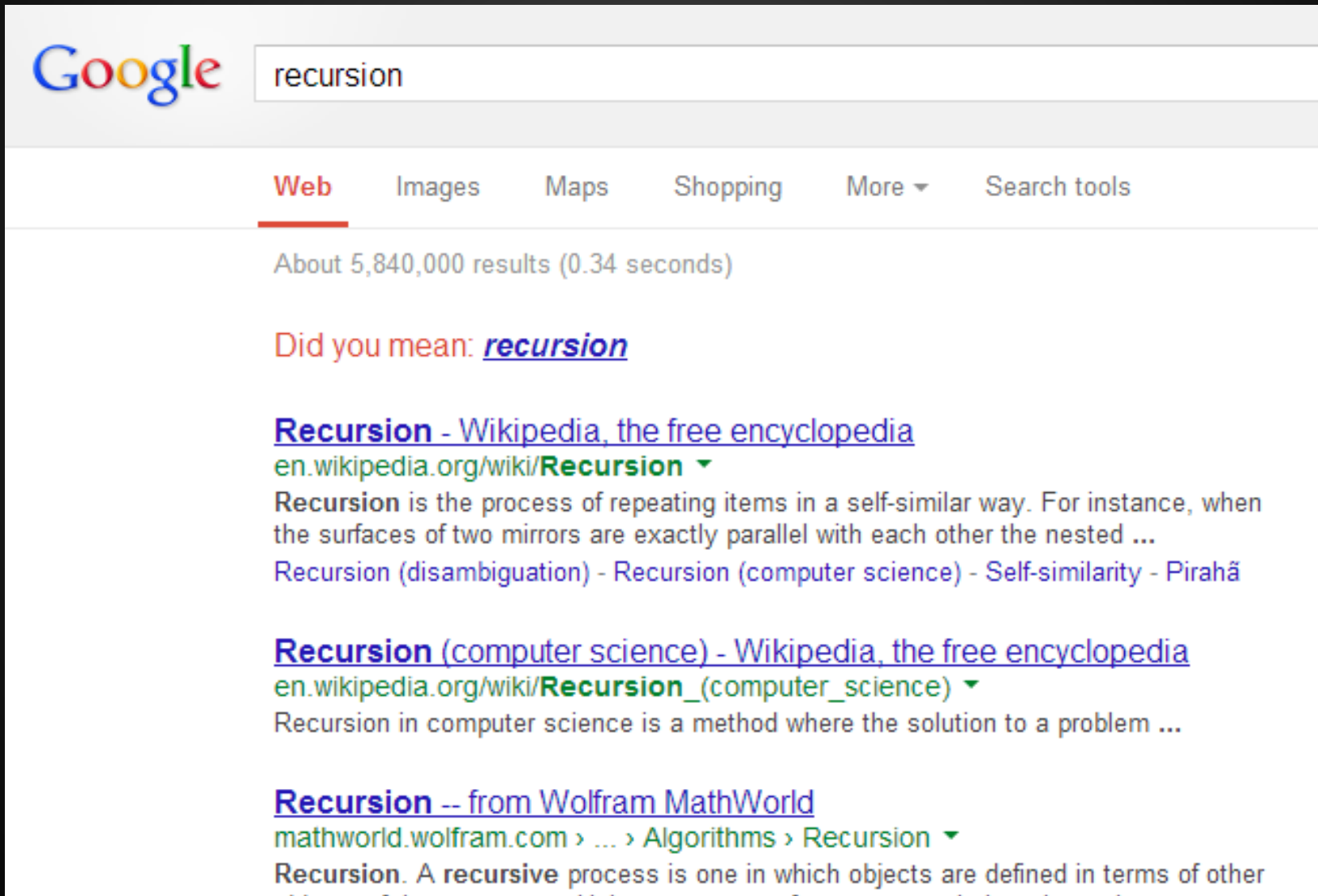
What is recursion?

Recursion is the process of defining an object or idea in terms of itself or something similar to itself



"In order to understand recursion, you must first understand recursion." - Urban Dictionary

What is recursion?



A screenshot of a Google search interface. The Google logo is on the left, and the search bar contains the word "recursion". Below the search bar, there are tabs for "Web", "Images", "Maps", "Shopping", "More", and "Search tools". The "Web" tab is selected. Below the tabs, it says "About 5,840,000 results (0.34 seconds)". A suggestion "Did you mean: [recursion](#)" is shown. Below that, there are three search results, each with a title, a URL, and a snippet.

Google

recursion

Web Images Maps Shopping More Search tools

About 5,840,000 results (0.34 seconds)

Did you mean: [recursion](#)

[Recursion - Wikipedia, the free encyclopedia](#)
en.wikipedia.org/wiki/Recursion ▼
Recursion is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other the nested ...
[Recursion \(disambiguation\)](#) - [Recursion \(computer science\)](#) - [Self-similarity](#) - [Pirahã](#)

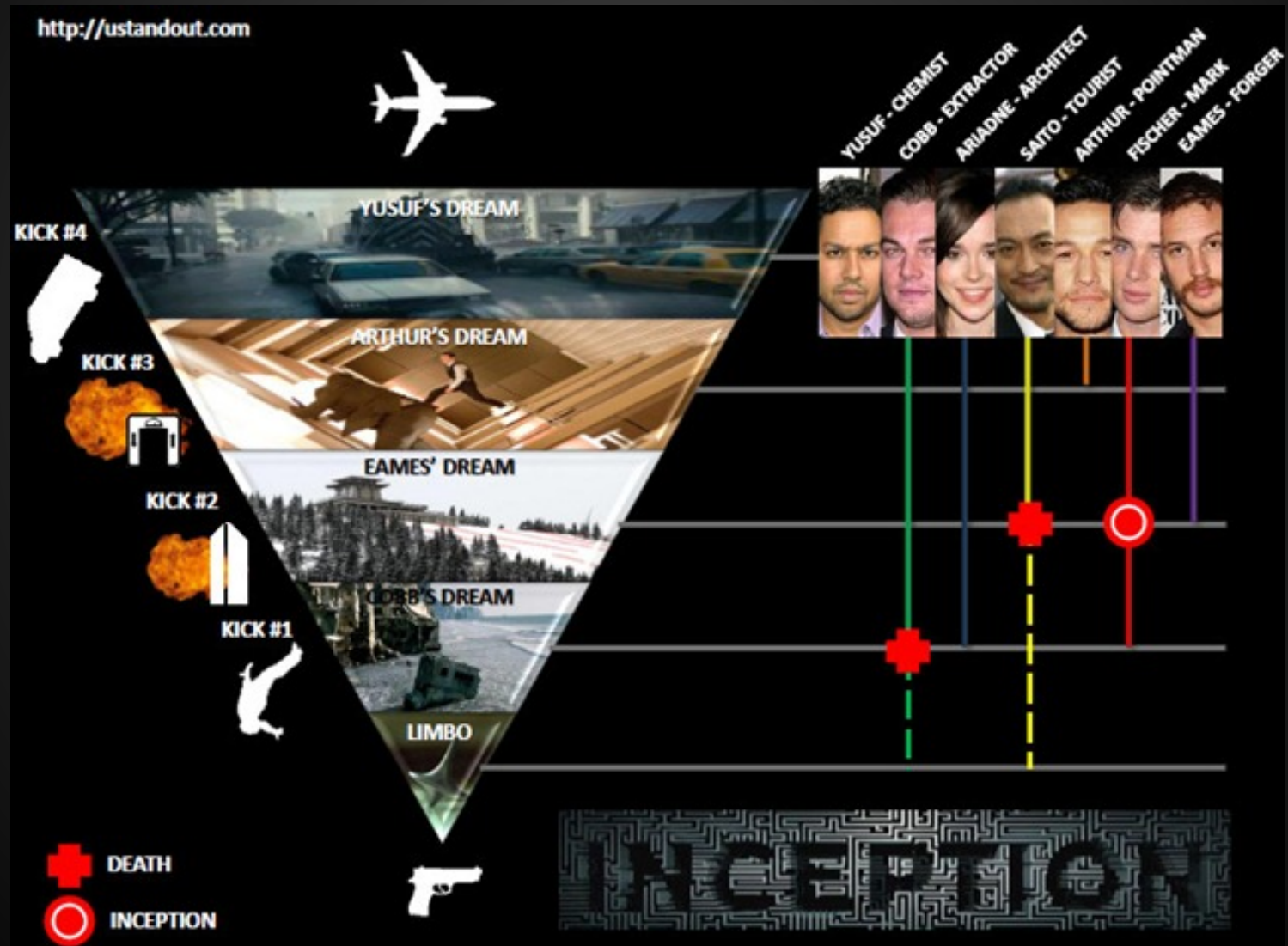
[Recursion \(computer science\) - Wikipedia, the free encyclopedia](#)
[en.wikipedia.org/wiki/Recursion_\(computer_science\)](http://en.wikipedia.org/wiki/Recursion_(computer_science)) ▼
Recursion in computer science is a method where the solution to a problem ...

[Recursion -- from Wolfram MathWorld](#)
mathworld.wolfram.com > ... > [Algorithms](#) > [Recursion](#) ▼
Recursion. A recursive process is one in which objects are defined in terms of other ...

What are some examples of recursion?



What are some examples of recursion?



What are some examples of recursion?

- Factorial

- $5!$
- $= 5 \times 4 \times 3 \times 2 \times 1$
- $= 5 \times (4 \times 3 \times 2 \times 1)$
- $= 5 \times 4!$

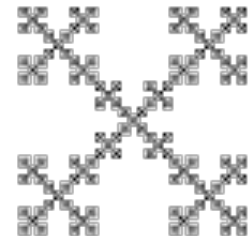
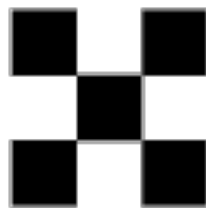
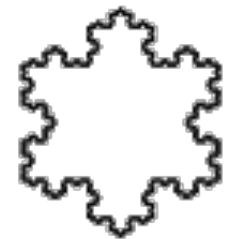
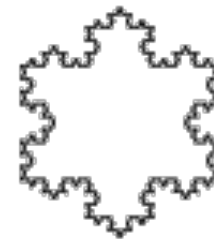
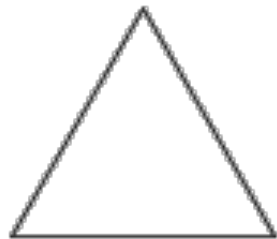
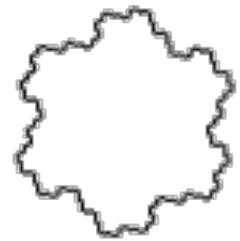
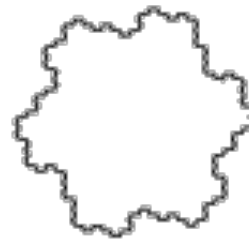
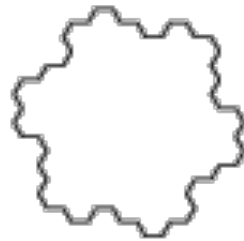
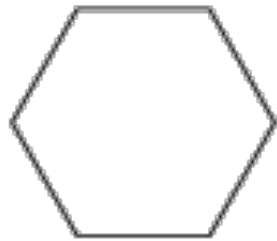
- Fibonacci sequence

- $2 = 1 + 1, 3 = 2 + 1, 5 = 3 + 2, 8 = 5 + 3, 13 = 8 + 5,$
...

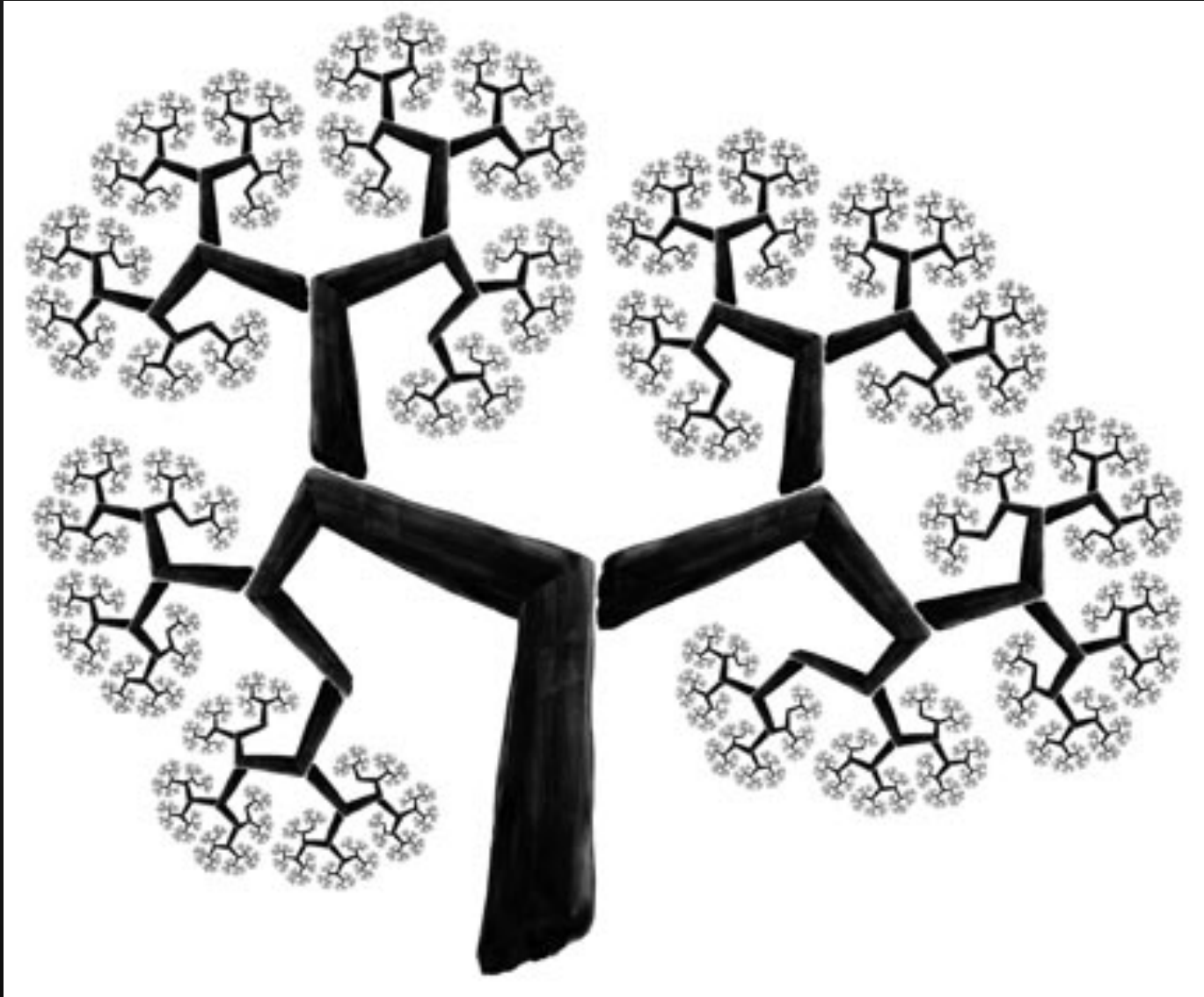
- Fractals

- images built upon recurring patterns at different scales

What do fractals look like?



What do fractals look like?

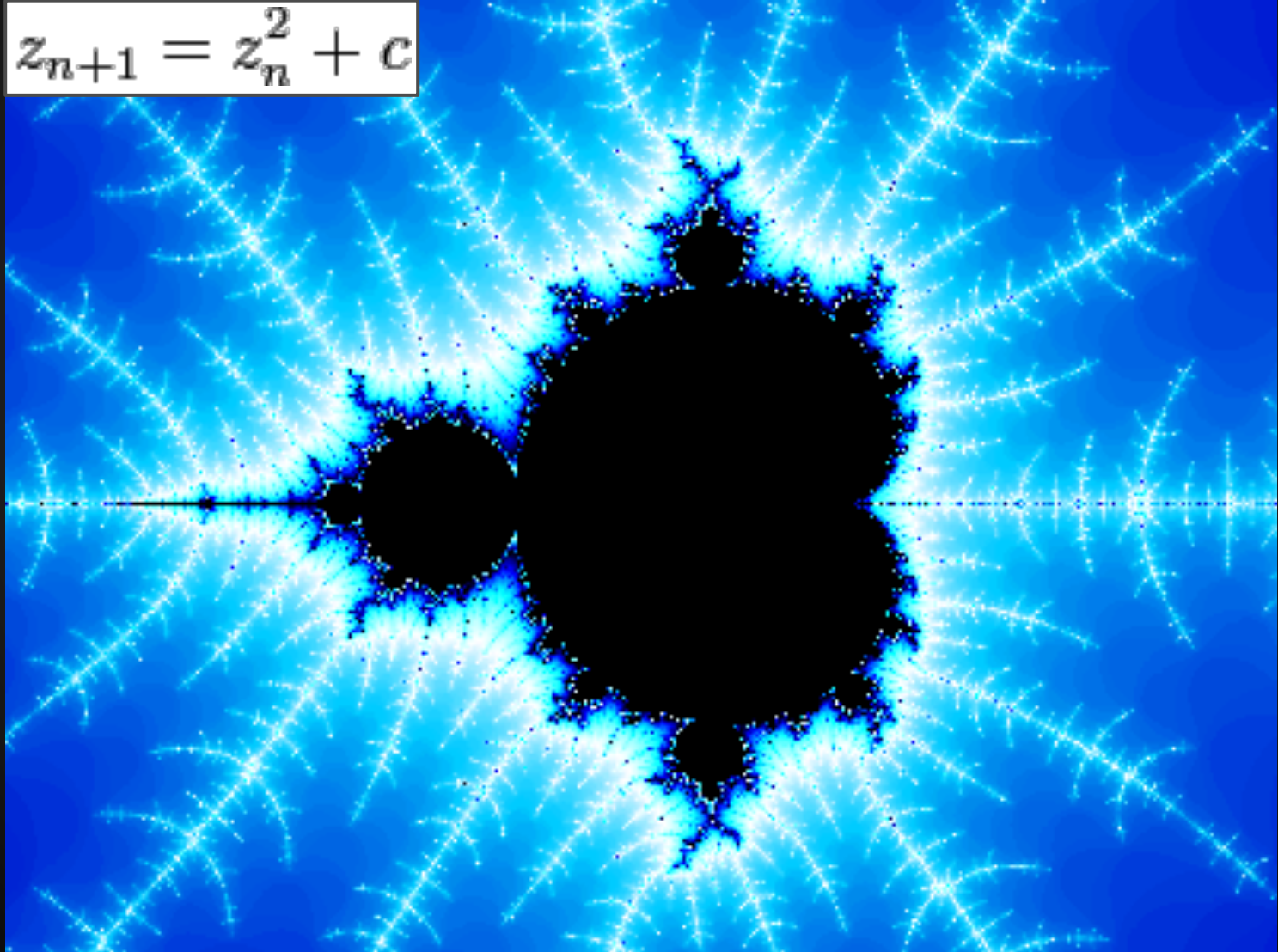


What do fractals look like?



What do fractals look like?

$$z_{n+1} = z_n^2 + c$$



Programming Recursion

How can you program recursion?

```
def foo(n):  
    print "foo is being called with ", n  
    foo(n-1)
```

What do you think will happen if you call foo?

foo will call foo, which will call foo, which will call foo, ... etc.

Infinite recursion

```
def foo(n):  
    print "foo is being called with ", n  
    foo(n-1)
```

```
foo(2)
```

```
"foo is being called with 2"
```

```
"foo is being called with 1"
```

```
"foo is being called with 0"
```

```
"foo is being called with -1"
```


```
"foo is being called with -2"
```

```
...
```

```
RangeError: Maximum call stack size exceeded
```

The base case

The smallest case(s) of a recursive function which are not defined in terms of themselves

```
def foo(n):  
    if n == 0:  
 print 'This is the base case of foo'  
    else:  
        print "foo is being called with ", n  
        foo(n-1)
```

The base case

```
def foo(n):  
    if n == 0:  
        print 'This is the base case of foo'  
    else:  
        print "foo is being called with ", n  
        foo(n-1)
```

```
foo(2)
```

```
foo is being called with 2  
foo is being called with 1  
This is the base case of foo
```

How do you program recursively?

1. THINK BEFORE YOU CODE
2. What is the smallest possible action? (the base case)
3. How can you create bigger actions by combining smaller ones? (the recursion)
4. Write your code for foo assuming that the recursive call to foo works

If you think about the recursive case while you are writing the recursive case, your head will explode

Programming factorial

THINK BEFORE YOU CODE

$$\begin{array}{lcl} 1! & = & 1 \\ 2! & = & 2 \times 1 \\ 3! & = & 3 \times 2 \times 1 \\ 4! & = & 4 \times 3 \times 2 \times 1 \\ 5! & = & 5 \times 4 \times 3 \times 2 \times 1 \\ 6! & = & 6 \times 5 \times 4 \times 3 \times 2 \times 1 \\ 7! & = & 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 \\ 8! & = & 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 \\ 9! & = & 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 \end{array}$$

What's the smallest possible action?

$$1! = 1$$

```
def factorial(n):  
    if n == 1:  
        return 1  
    ...
```

How can you create bigger actions by combining smaller ones?

$$1! = 1$$

$$2! = 2 \times 1!$$

$$3! = 3 \times 2!$$

$$4! = 4 \times 3!$$

$$5! = 5 \times 4!$$

$$6! = 6 \times 5!$$

...

$$n! = n \times (n-1)!$$

How can you create bigger actions by combining smaller ones?

$$n! = n \times (n-1)!$$

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n-1)
```

Write your code for factorial assuming factorial works

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n-1)
```



**Assume this
works**

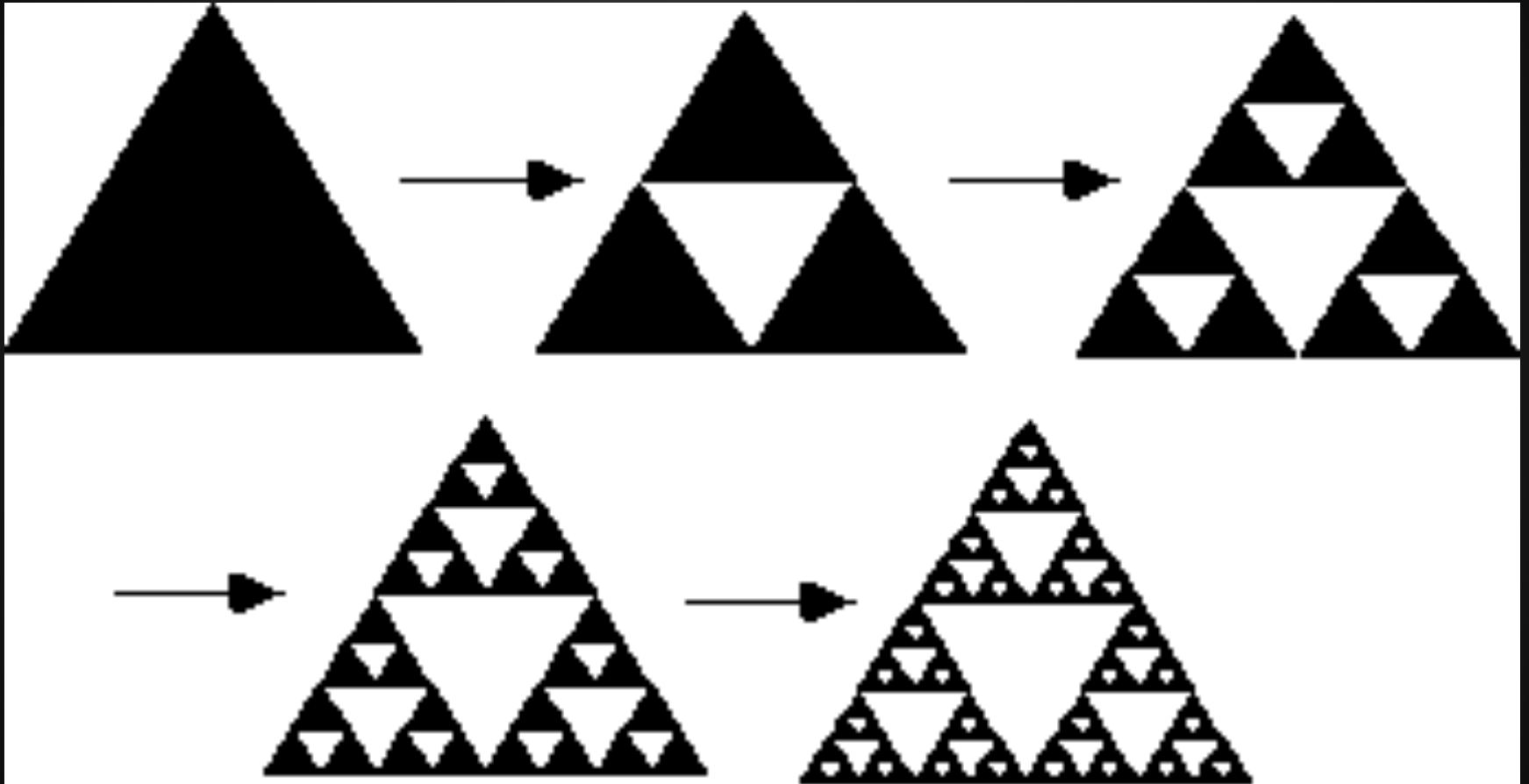
How your computer runs factorial

```
factorial(5)
= 5 * factorial(4)
= 5 * (4 * factorial(3))
= 5 * (4 * (3 * factorial(2)))
= 5 * (4 * (3 * (2 * factorial(1))))
= 5 * (4 * (3 * (2 * (1))))
= 5 * (4 * (3 * (2 * 1)))
= 5 * (4 * (3 * 2))
= 5 * (4 * 6)
= 5 * (24)
= 120
```

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

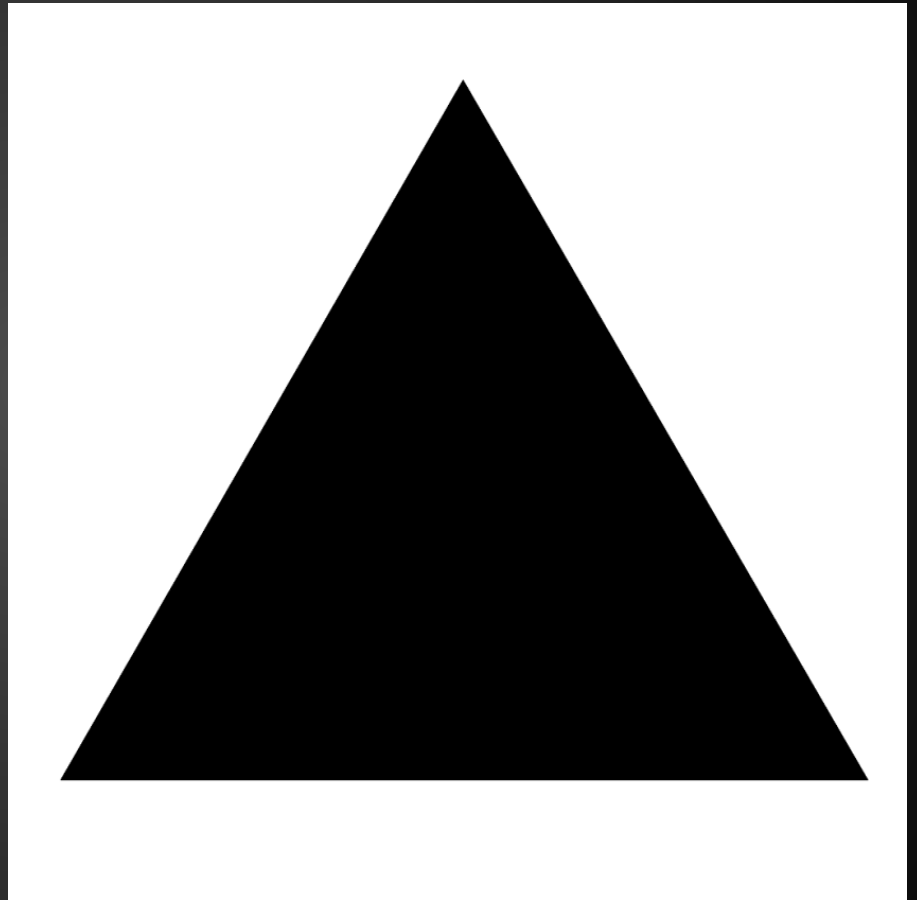
Programming Sierpinski Triangle

THINK BEFORE YOU CODE



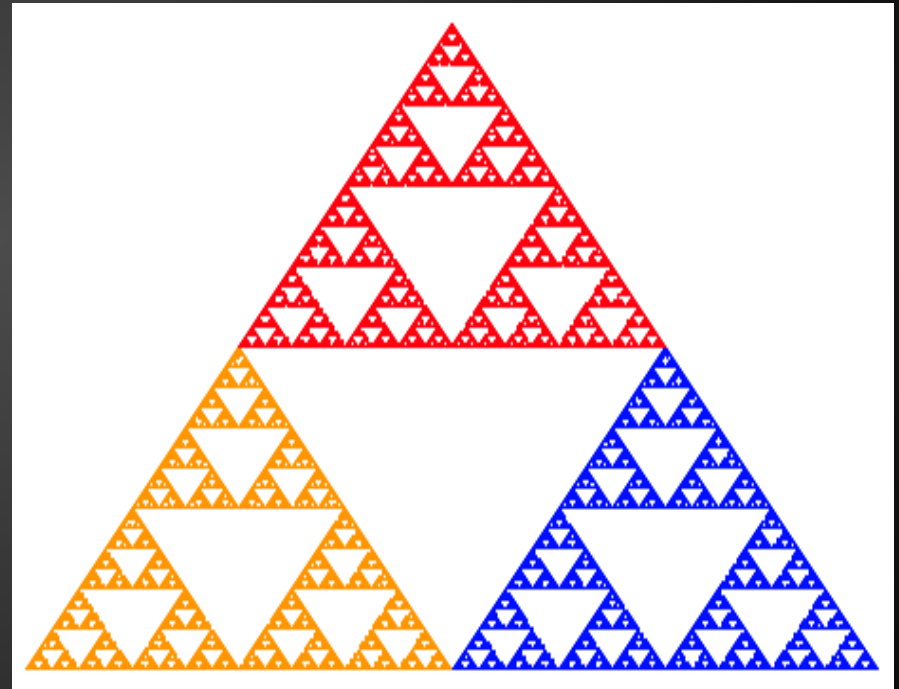
What's the smallest possible action?

```
def sierpinski(size):  
    if size <= 20:  
        triangle(size)  
    ...
```



How can you create bigger actions by combining smaller ones?

```
def sierpinski(size):  
    if size <= 20:  
        triangle(size)  
    else:  
        sierpinski(size/2)  
        winston.forward(size/2)  
        sierpinski(size/2)  
        winston.backward(size/2)  
        winston.left(60)  
        winston.forward(size/2)  
        winston.right(60)  
        sierpinski(size/2)  
        winston.left(60)  
        winston.backward(size/2)  
        winston.right(60)
```



Write your code for `sierpinski` assuming `sierpinski` works

```
def sierpinski(size):  
    if size <= 20:  
        triangle(size)  
    else:  
        sierpinski(size/2)  
        winston.forward(size/2)  
        sierpinski(size/2)  
        winston.backward(size/2)  
        winston.left(60)  
        winston.forward(size/2)  
        winston.right(60)  
        sierpinski(size/2)  
        winston.left(60)  
        winston.backward(size/2)  
        winston.right(60)
```



**Assume
these work**

WTF, why is this useful?

- Code is usually more concise and can be "elegant"
- Many applications are easiest to program recursively
- Because it's cool
- Do you need more reasons?

Questions?

Another Pop Quiz!

