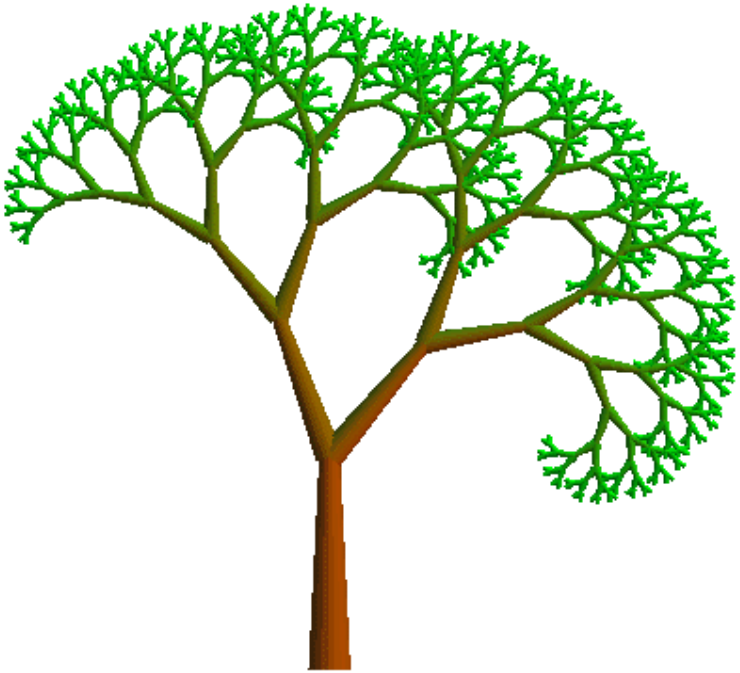
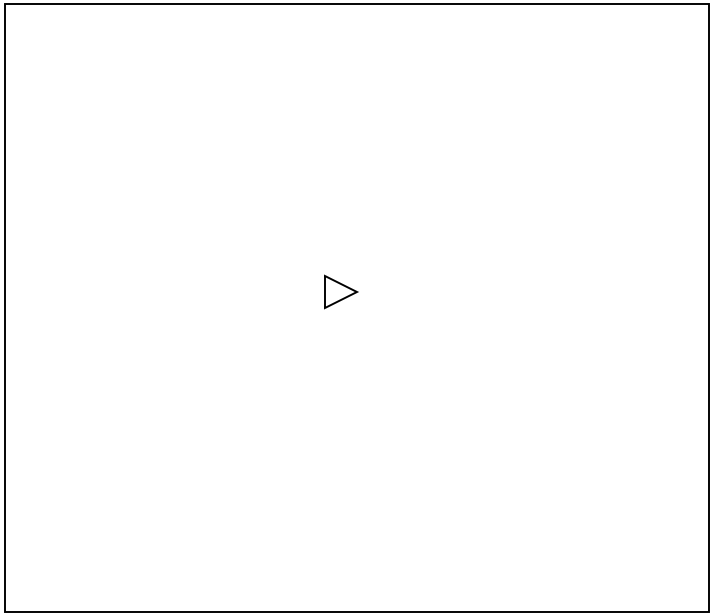
Dictionaries and Map Paths



Recursive pictures
Tuples: immutable sequences
Dictionaries: a new data type
Using a dictionary to compute routes in a map

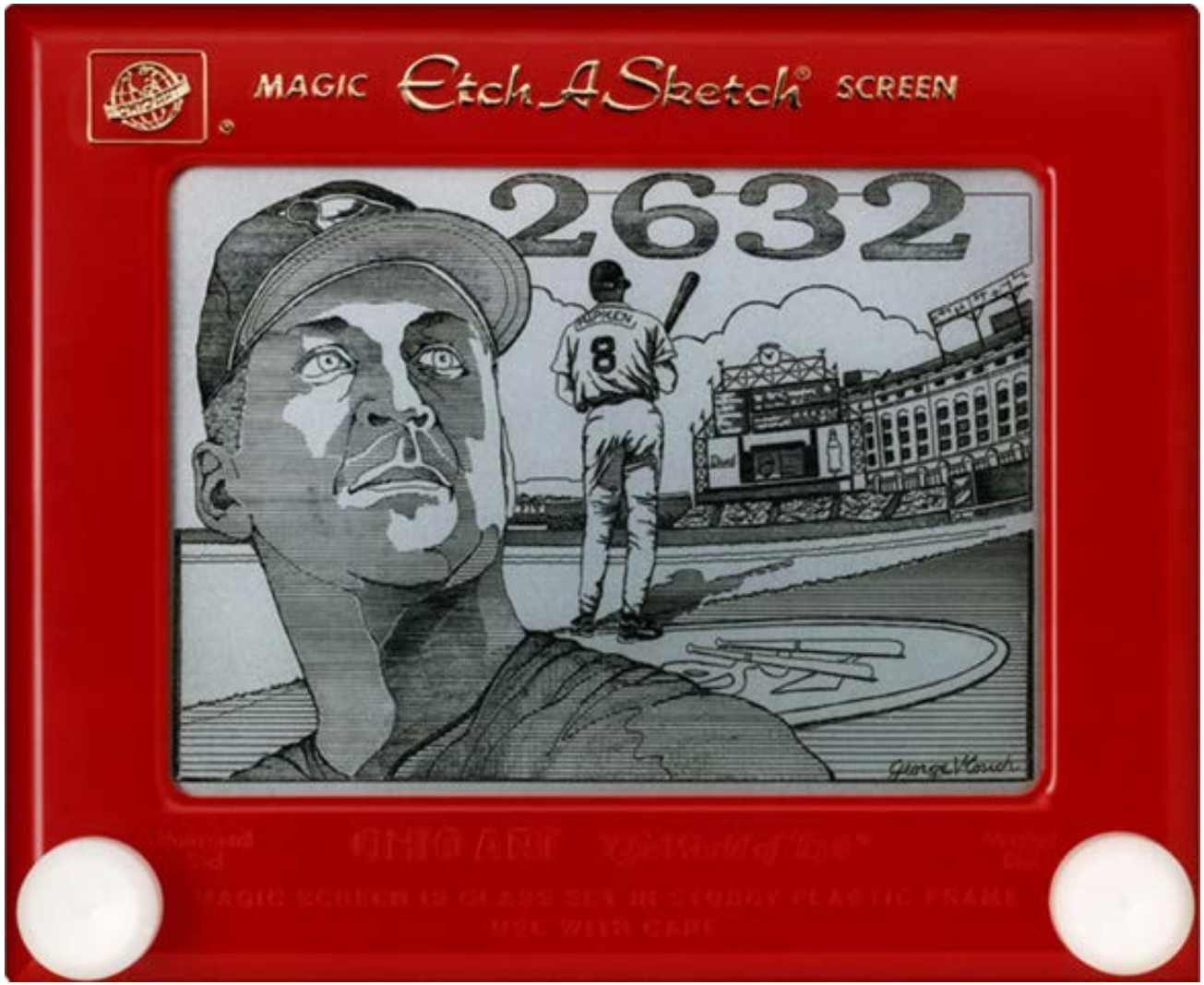
Turtle Graphics



forward(100)

right(90)

Etch-a-Sketch craziness…



No way this is real…

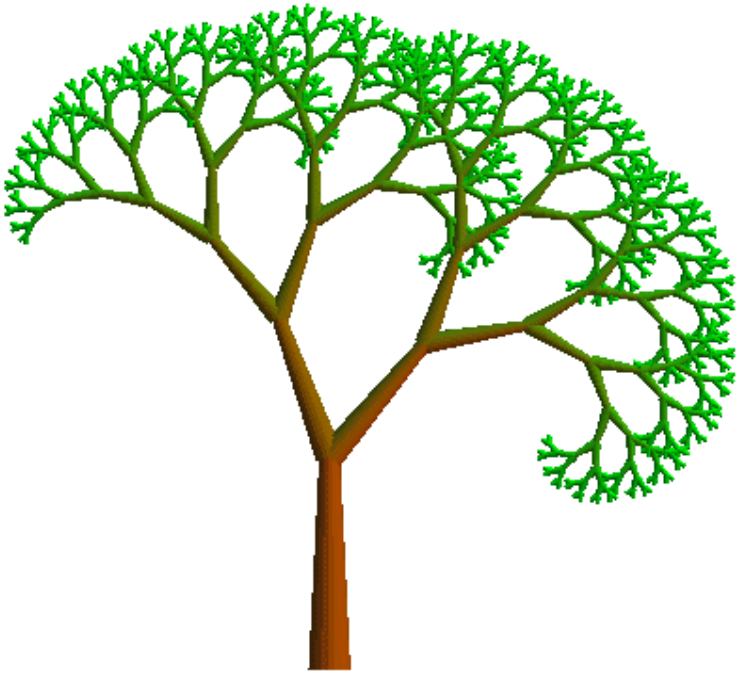
except that it is !

Turtle Graphics

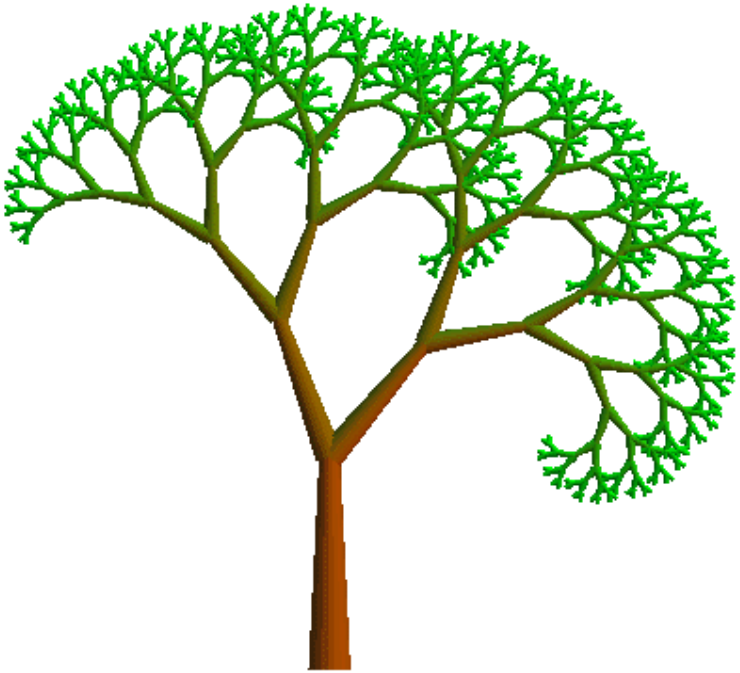
```
>>> import turtle
>>> turtle.forward(50)
>>> turtle.right(90)
>>> turte.backward(50)
```

degrees()	radians()	reset()
clear()	tracer(flag)	forward(distance)
backward(distance)	left(angle)	right(angle)
up()	down()	width(width)
color(*args)	fill(flag)	heading()
setheading(angle)	window_width()	window_height()
position()	setx(xpos)	sety(ypos)
goto(x,y)		

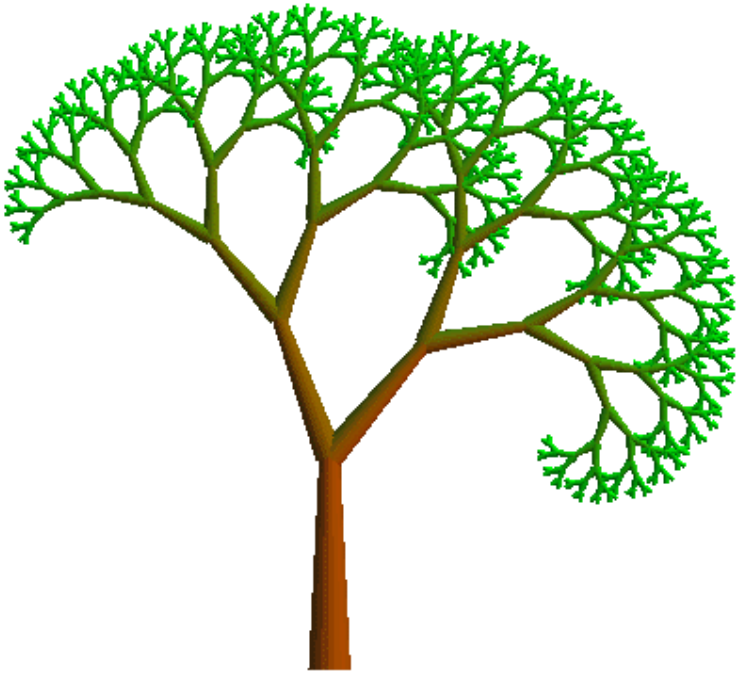
Fractals



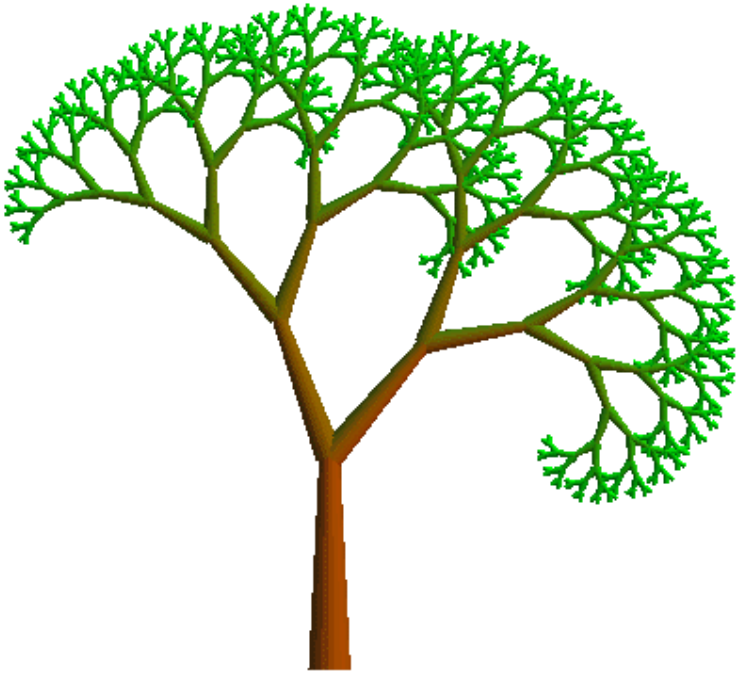
Fractals



Fractals



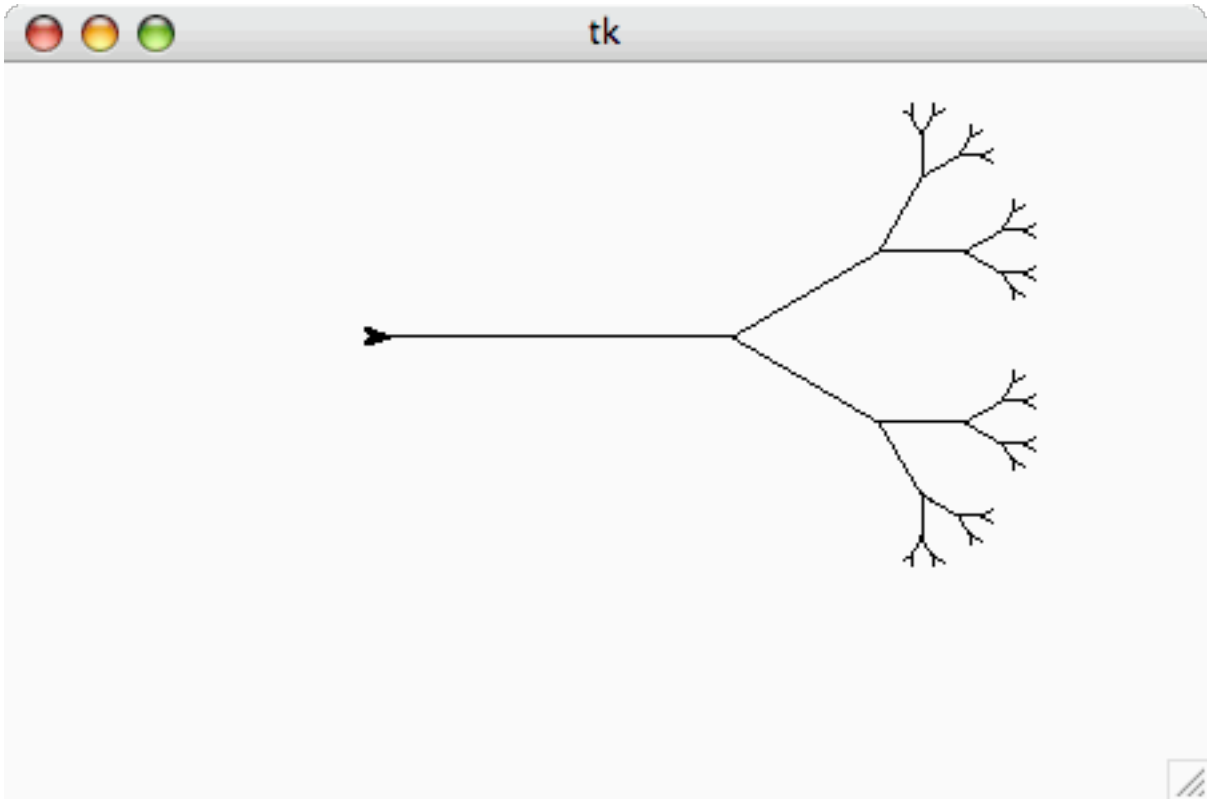
Fractals



“I wonder about Trees” – Robert Frost

“We wonder about Robert Frost” - Trees

```
>>> svTree(128, 6)
```



recursion level trunk length

```
>>> svTree(3, 100)
```

recursion level trunk length

```
>>> svTree(3, 100)
```

100 long!

recursion level trunk length

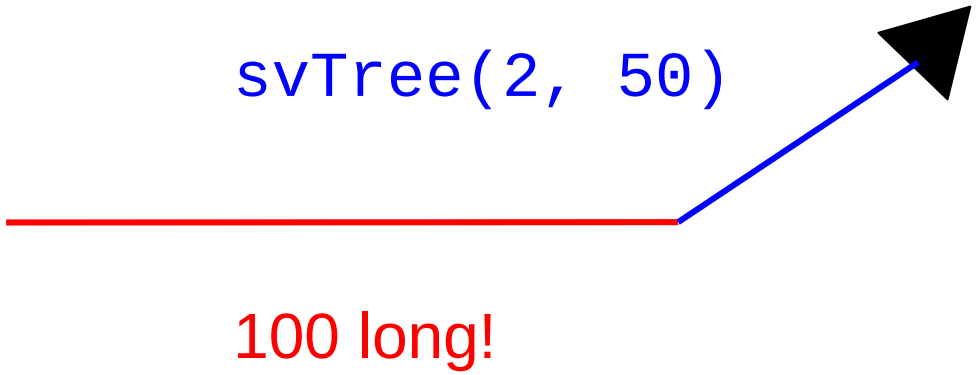
```
>>> svTree(3, 100)
```

```
svTree(2, 50)
```

100 long!

recursion level trunk length

```
>>> svTree(3, 100)
```

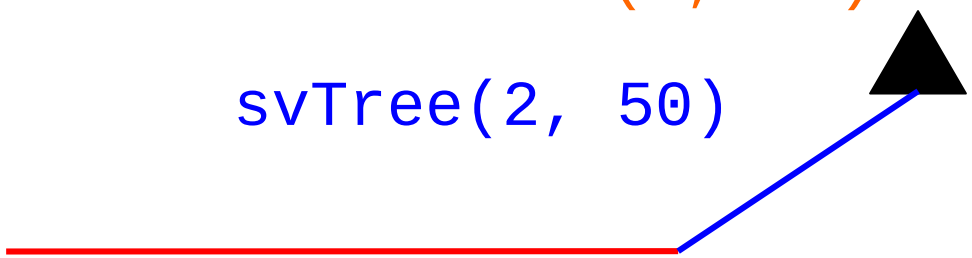


recursion level trunk length

```
>>> svTree(3, 100)
```

svTree(1, 25)

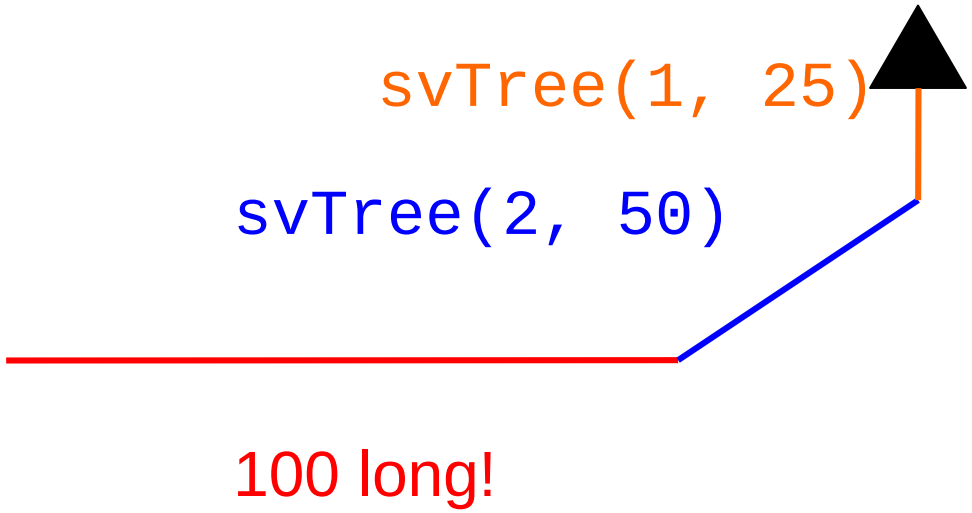
svTree(2, 50)



100 long!

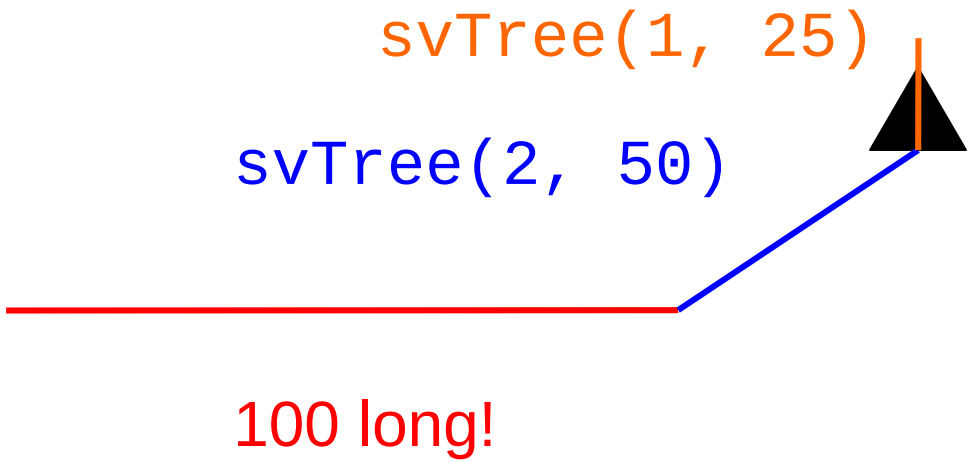
recursion level trunk length

```
>>> svTree(3, 100)
```



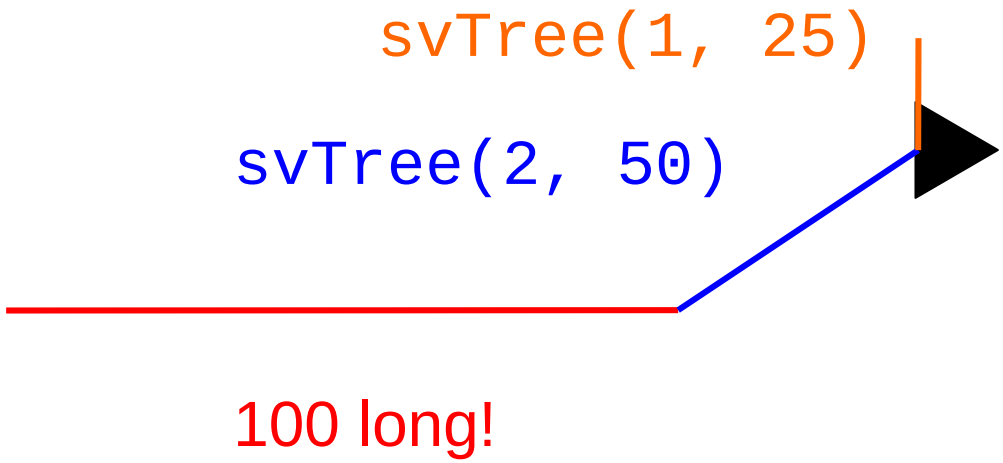
recursion level trunk length

```
>>> svTree(3, 100)
```



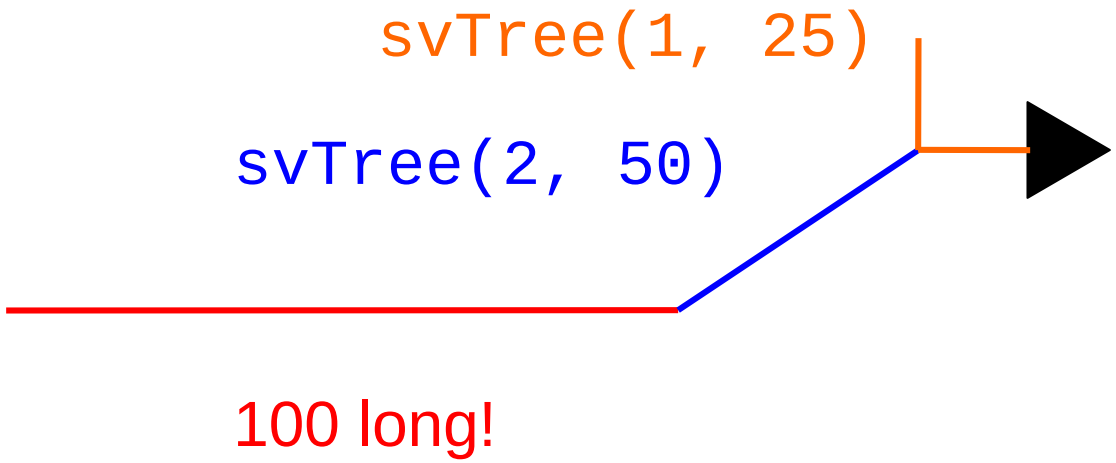
recursion level trunk length

```
>>> svTree(3, 100)
```



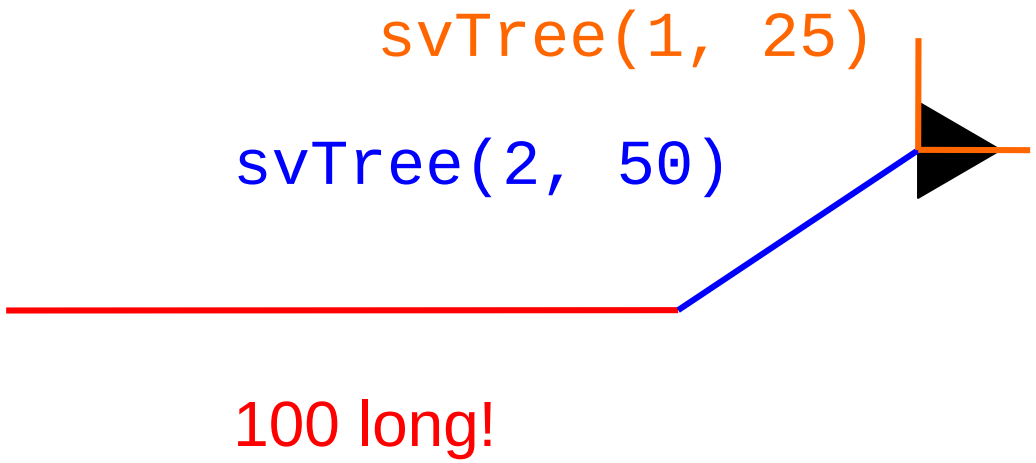
recursion level trunk length

```
>>> svTree(3, 100)
```



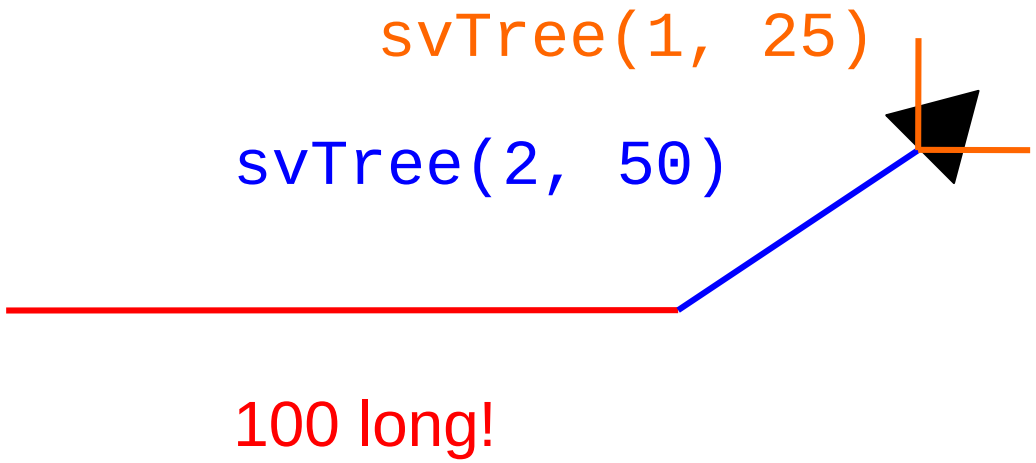
recursion level trunk length

```
>>> svTree(3, 100)
```



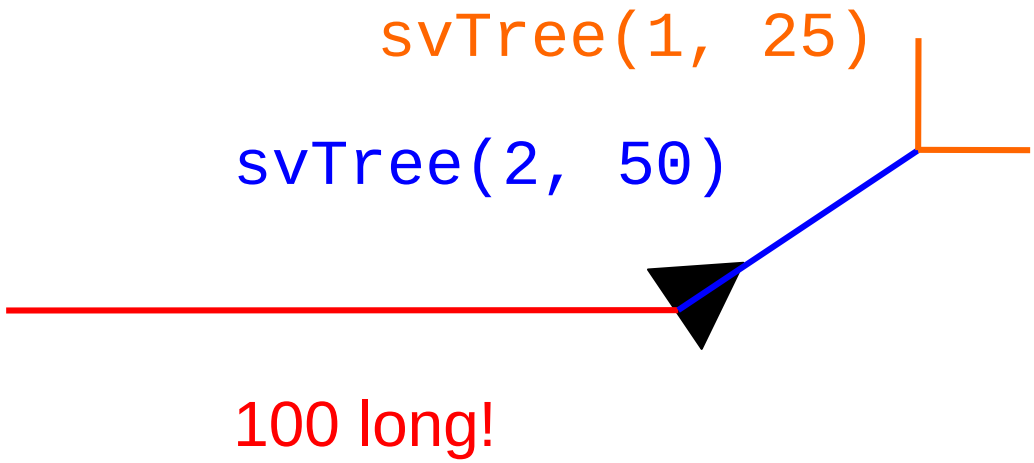
recursion level trunk length

```
>>> svTree(3, 100)
```



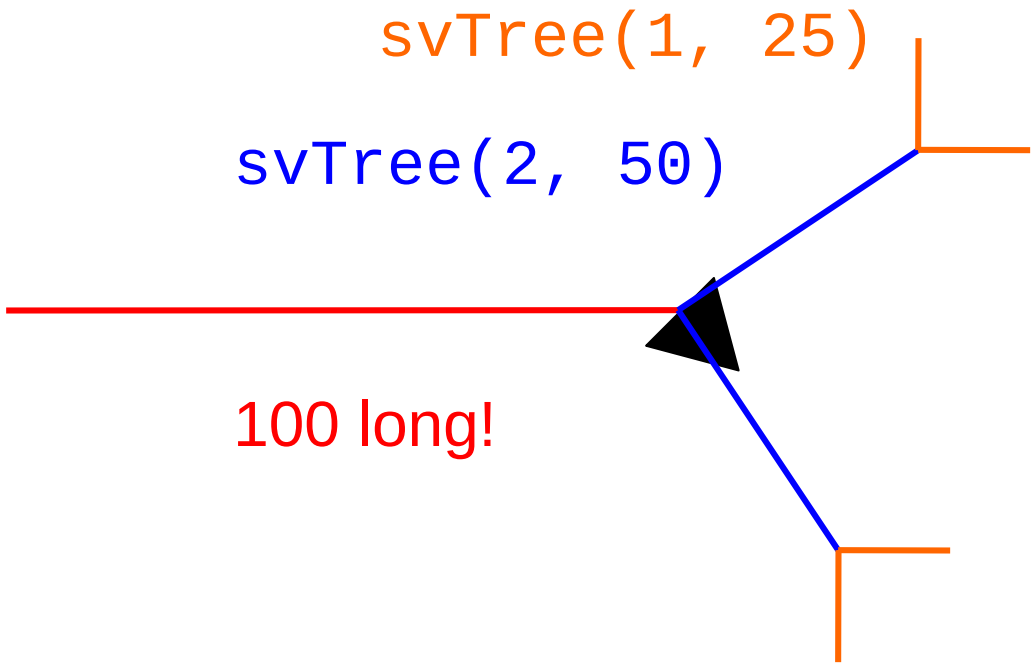
recursion level trunk length

```
>>> svTree(3, 100)
```



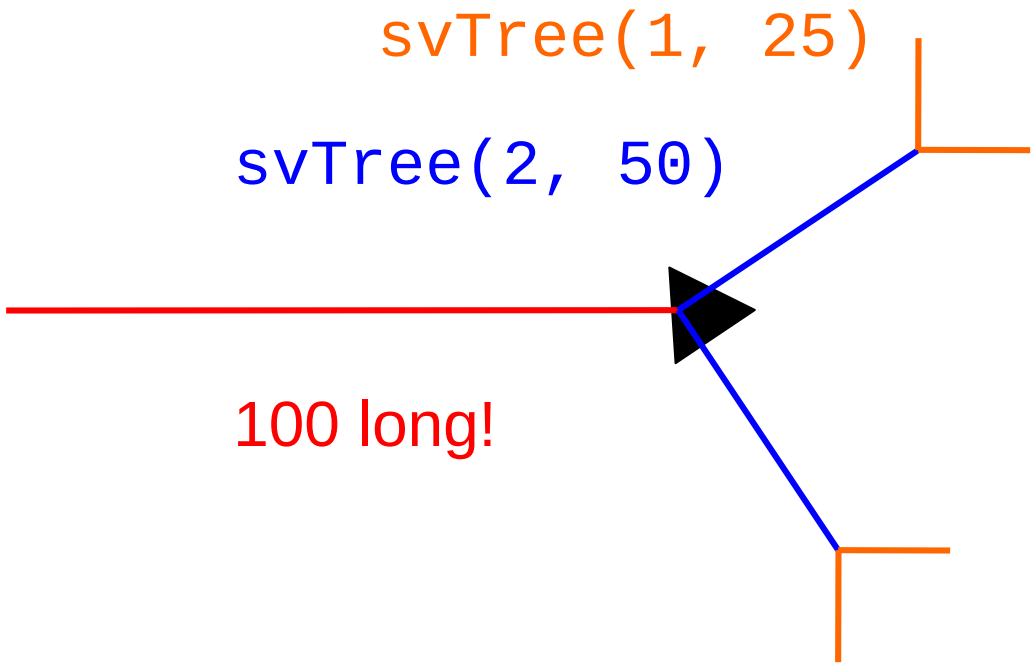
recursion level trunk length

```
>>> svTree(3, 100)
```



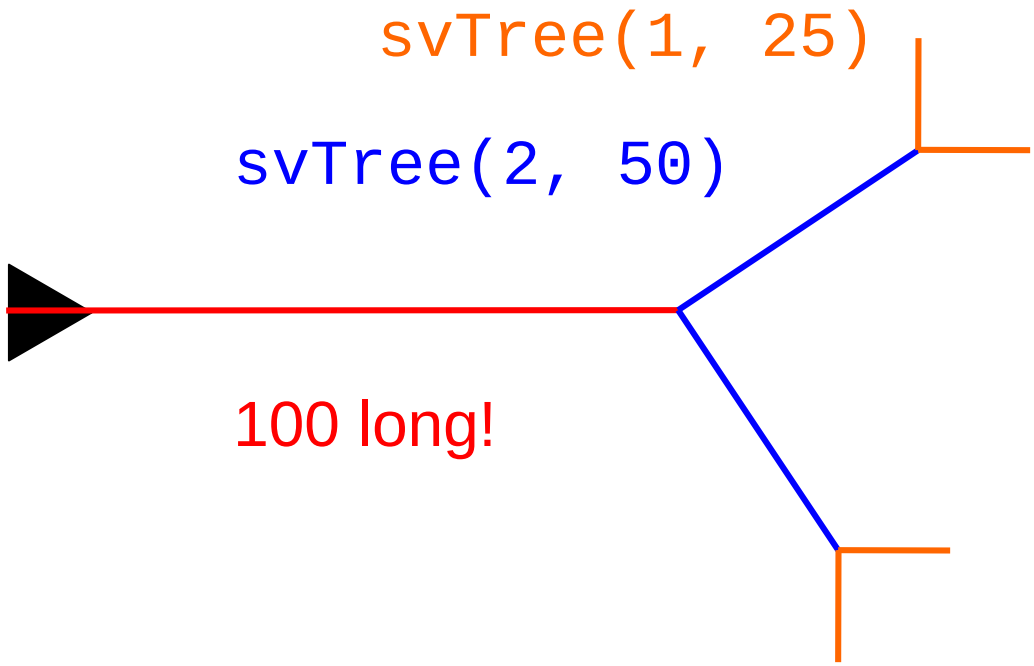
recursion level trunk length

```
>>> svTree(3, 100)
```



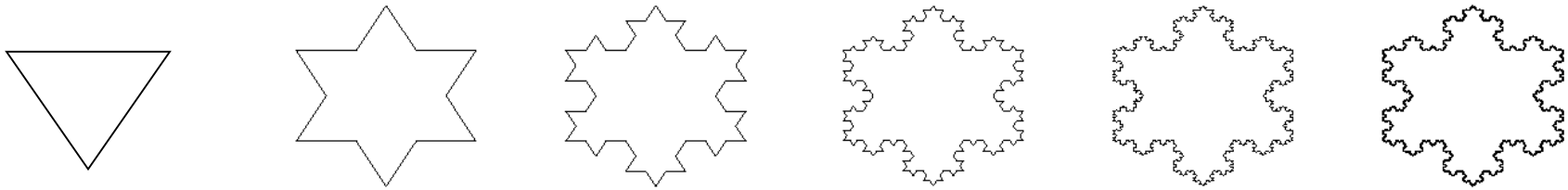
recursion level trunk length

```
>>> svTree(3, 100)
```



SnowFlake Fractals

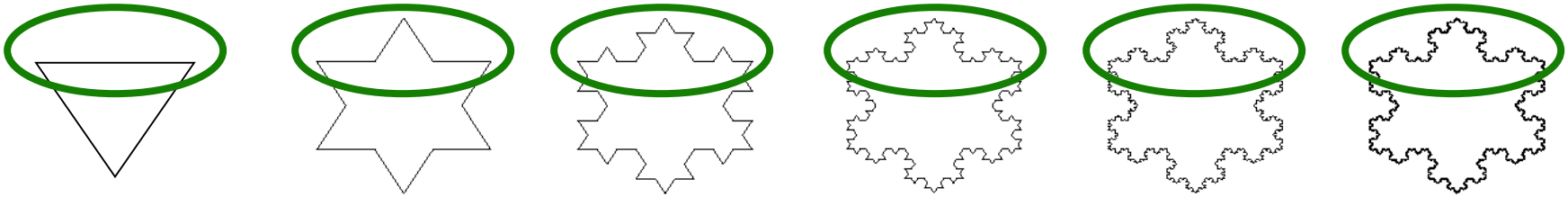
The Koch Snowflake Fractal:



level 0 level 1 level 2 level 3 level 4 level 5

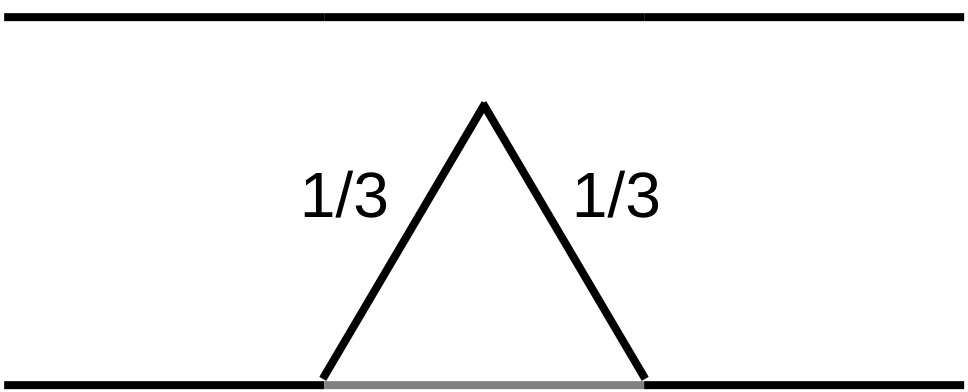
Snowflake Fractals

The Koch Snowflake Fractal:



level 0 level 1 level 2 level 3 level 4 level 5

level 0



level 1

1/3 1/3 1/3

And now for something completely different…

(…or at least it will seem different until we see that it's not!)

Tuples (“immutable lists”)

```
>>> foo = (42, 'hello', (5, 'spam'), 'penguin')
>>> foo
(42, 'hello', (5, 'spam'), 'penguin')
>>> foo[0]
42
>>> foo[-1]
'penguin'
>>> foo[0:2]
(42, 'hello')
>>> foo[0:1]
(42,)
```

Tuples (“immutable lists”)

```
>>> foo = (42, 'hello', (5, 'spam'), 'penguin')
>>> foo
(42, 'hello', (5, 'spam'), 'penguin')
>>> foo[0]
42
>>> foo[-1]
'penguin'
>>> foo[0:2]
(42, 'hello')
>>> foo[0:1]
(42,)
>>> foo[0] = 100
BARF!!! (that’s California-speak for ‘error’)
```

Dictionaries

```
>>> D = {}
>>> D[“Ran”]= “spam”
>>> D[“Zach”]= “donuts”
>>> D[“Alien”]= 42
>>> D[“Ran”]
'spam'
>>> D[“Alien”]
42
>>> D[“Napoleon Dynamite”]
BARF!
```

“Ran”, “Zach”, and “Alien” are called the “keys” in the dictionary. Any ***immutable*** object can be a key.

Dictionaries

```
>>> D = {}
>>> D[“Ran”]= “spam”
>>> D[“Zach”]= “donuts”
>>> D[“Alien”]= 42
>>> D[“Ran”]
'spam'
>>> D[“Alien”]
42
>>> D[“Napoleon Dynamite”]
BARF!
>>> D
{ ‘Ran’: ‘spam’, ‘Zach’: ‘donuts’, ‘Alien’ : 42 }
```

Dictionaries - summary

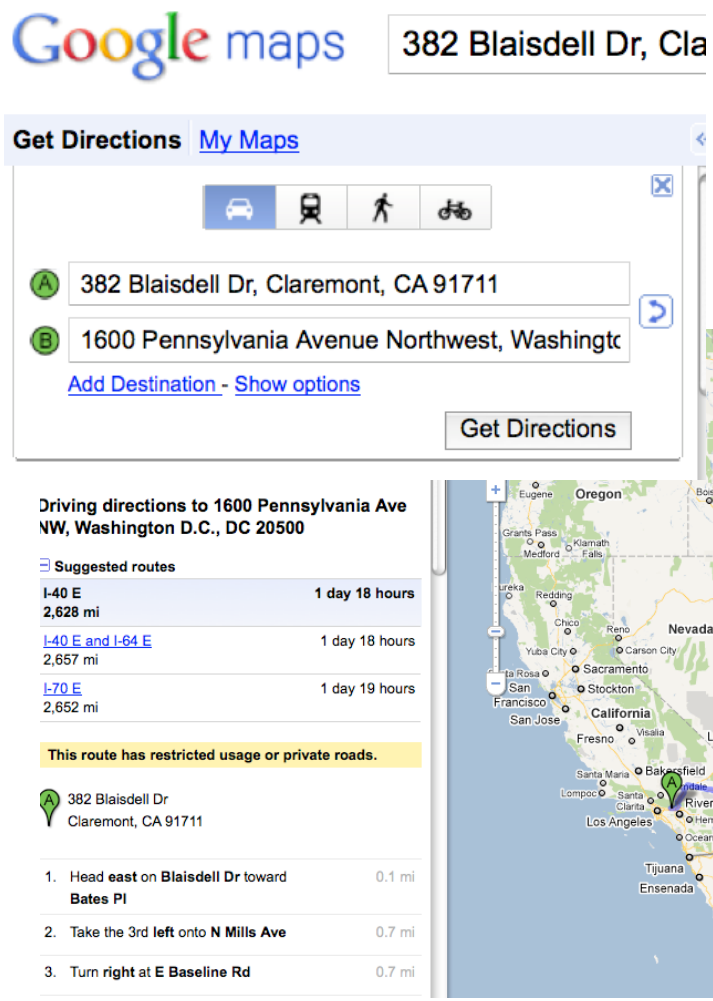
A dictionary associates values with keys.

```
D = {}   # create an empty dictionary
D[k]= v  # make key k have value v
         # (replace old value if k already in D)
D[k]     # get value under key k
D.has_key(k) # whether k is a key in D
```

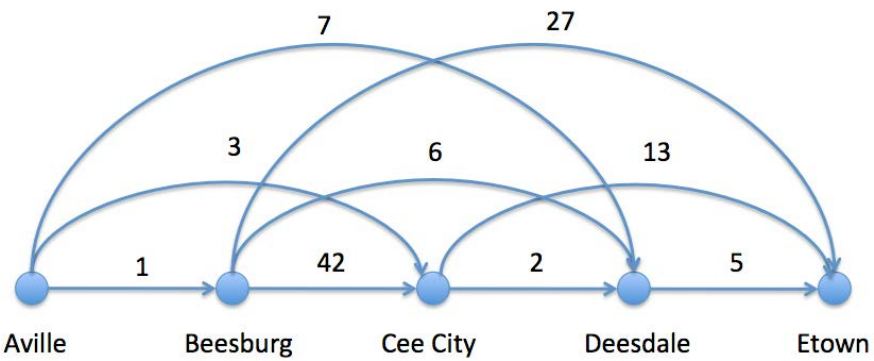
Example: {‘cat’:3, ‘avatar’:1, ‘sprite’:42}

v can be any value
k must be an immutable type (string, int, tuple of immutables)

Finding Shortest Paths



Giigle maps

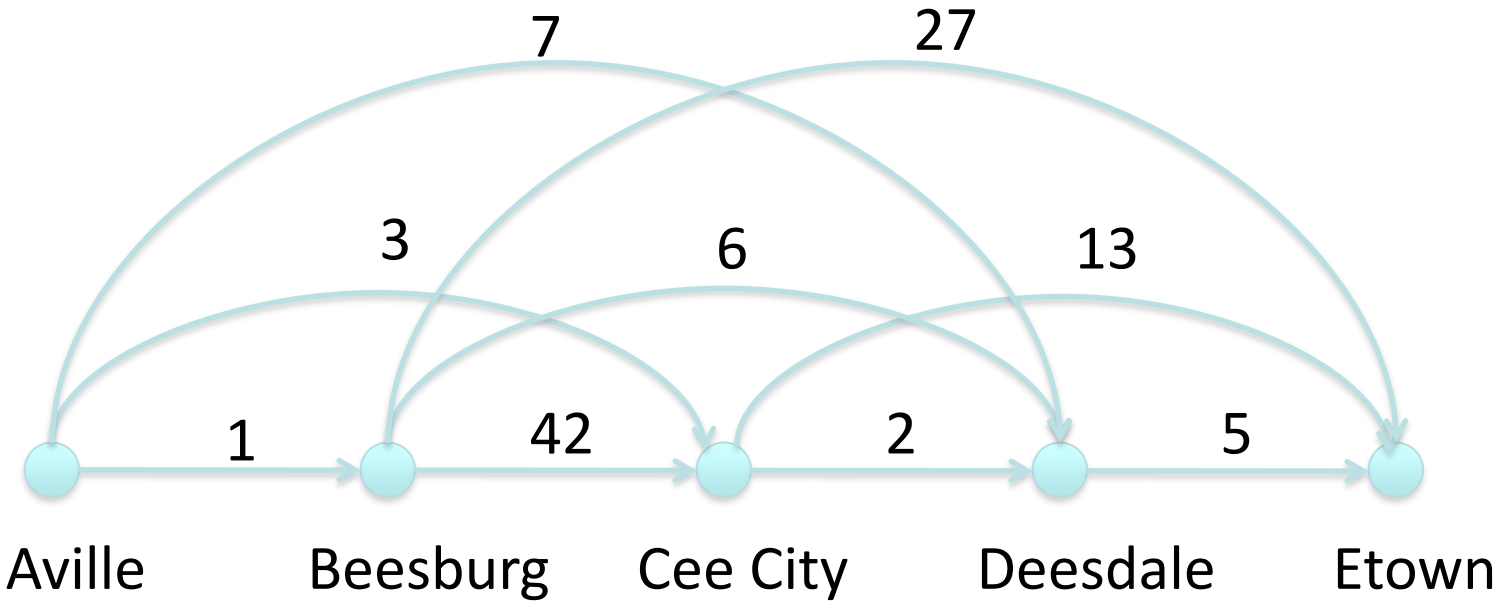



```
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists  = {("A","A"):0, ("A","B"):1, ("A","C"):3, ("A","D"):7 , ("A","E"):Inf,
              ("B","A"):Inf, ("B","B"):0, ("B","C"):42, ("B","D"):6, ("B","E"):27,
              ("C","A"):Inf, ("C","B"):Inf, ("C","C"):0, ("C","D"):2, ("C","E"):13,
              ("D","A"):Inf, ("D","B"):Inf, ("D","C"):Inf, ("D","D"):0, ("D","E"):5,
              ("E","A"):Inf, ("E","B"):Inf, ("E","C"):Inf, ("E","D"):Inf, ("E","E"):0
             }
```

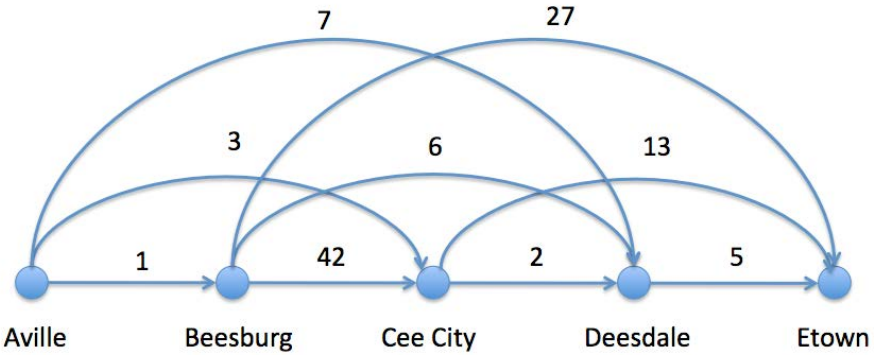
```
>>> FiveDists[ (“B”, “C”) ]
42
```

Sometimes we need to make more than 2 recursive calls!

Giigle maps

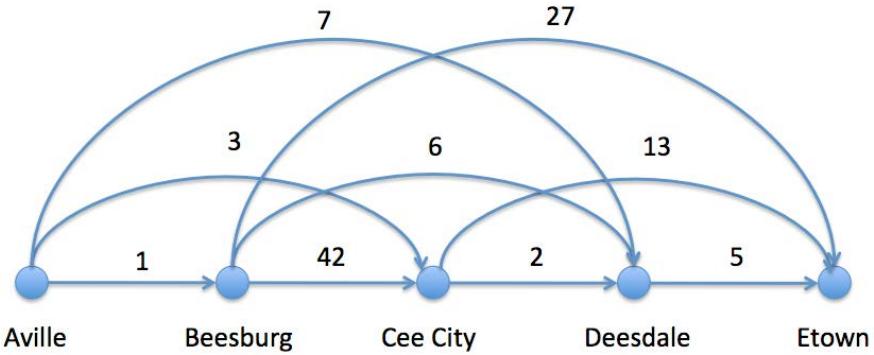


Shortest path?
Is greed good?
How does the use-it-or-lose-it idea get used here?




```
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists  = {("A","A"):0, ("A","B"):1, ("A","C"):3, ("A","D"):7 , ("A","E"):Inf,
              ("B","A"):Inf, ("B","B"):0, ("B","C"):42, ("B","D"):6, ("B","E"):27,
              ("C","A"):Inf, ("C","B"):Inf, ("C","C"):0, ("C","D"):2, ("C","E"):13,
              ("D","A"):Inf, ("D","B"):Inf, ("D","C"):Inf, ("D","D"):0, ("D","E"):5,
              ("E","A"):Inf, ("E","B"):Inf, ("E","C"):Inf, ("E","D"):Inf, ("E","E"):0
             }
```

```
>>> shortestPath (FiveCities, FiveDists)
10

>>> shortestPath (["C", "D", "E"], FiveDists)
7

>>> shortestPath (["E"], FiveDists)
0
```




```
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists  = {("A","A"):0, ("A","B"):1, ("A","C"):3, ("A","D"):7 , ("A","E"):Inf,
              ("B","A"):Inf, ("B","B"):0, ("B","C"):42, ("B","D"):6, ("B","E"):27,
              ("C","A"):Inf, ("C","B"):Inf, ("C","C"):0, ("C","D"):2, ("C","E"):13,
              ("D","A"):Inf, ("D","B"):Inf, ("D","C"):Inf, ("D","D"):0, ("D","E"):5,
              ("E","A"):Inf, ("E","B"):Inf, ("E","C"):Inf, ("E","D"):Inf, ("E","E"):0
             }
```

```
def shortestPath (Cities, Distances):
    ''' returns the length of the shortest path
    from the leftmost to the rightmost city in
    the Cities list.'''
    if BLAH:  return BLAH BLAH
    else: return BLAH BLAH BLAH
```

Just two lines of code!!!

It's fitting that map gets used here!

Google maps

Search Maps

Show search options

Get Directions My Maps

382 Blaisdell Dr, Claremont, CA 91711

23 Ronald Avenue, Greenwich, New South Wale

Add Destination - Show options

Get Directions

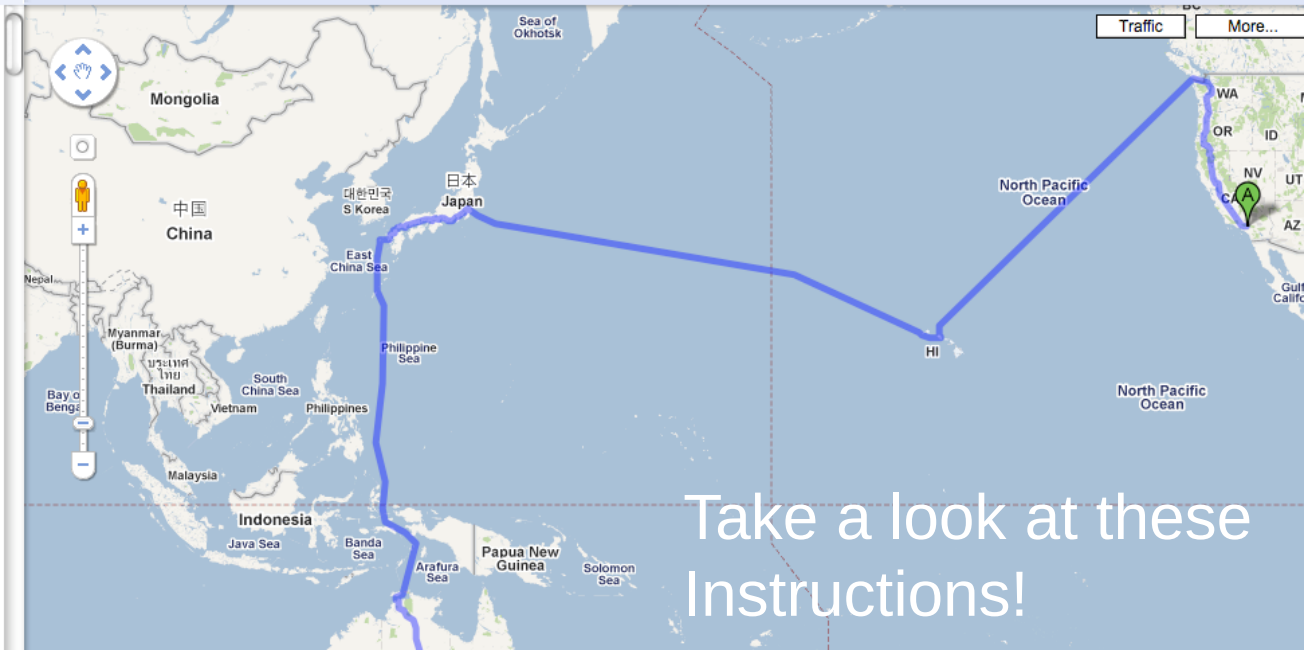
Driving directions to 23 Ronald Ave, Greenwich NSW 2065, Australia

I-5 N **55 days 6 hours**
14,535 mi

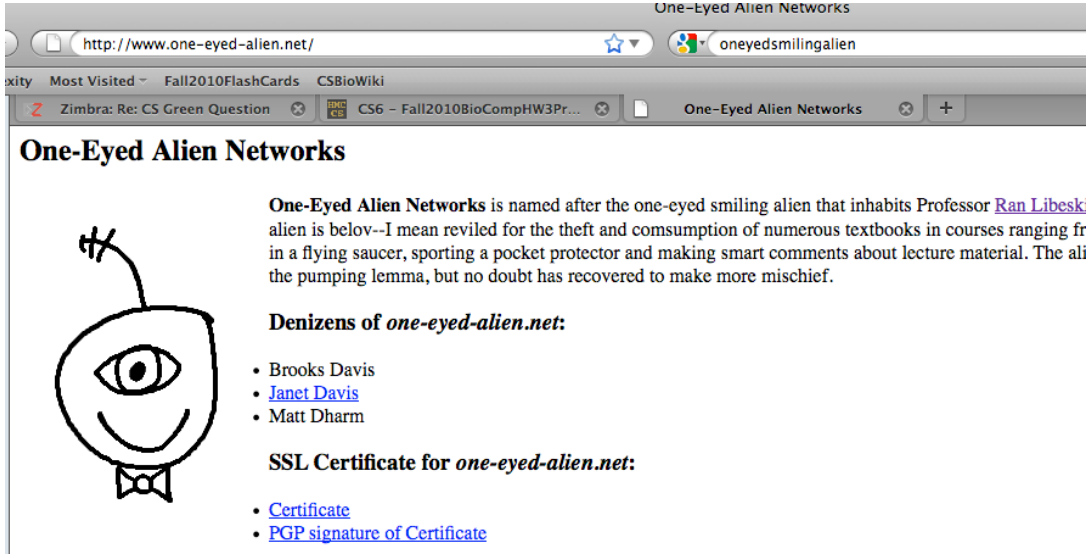
This route has tolls.
This route includes a ferry.
This route crosses through Japan.

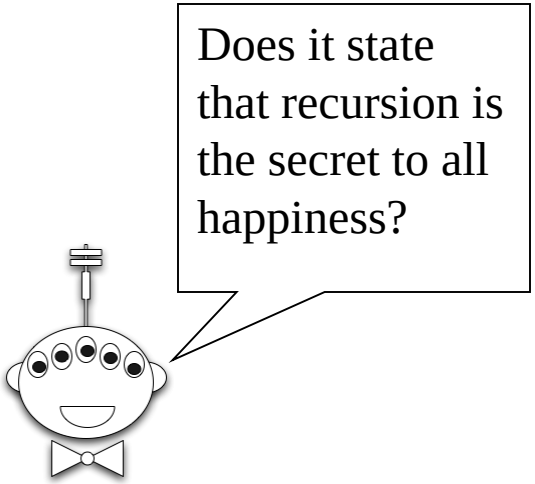
382 Blaisdell Dr
Claremont, CA 91711

1. Head **east** on **Blaisdell Dr** toward **Bates Pl** 0.1 mi



The Recursion Theorem





```
public class Quine{
public static void main(String [] args){
String t ="'/#"+
"                                   Str                                        #"+
"                                   ing                                        #"+
"                                  []                                          #"+
"                                 fstr                                         #"+
"                                 ing                                          #"+
"                                 s = t                                        #"+
"                               .spli                                          #"+
"                               t('@')                                         #"+
"                              ;String                                         #"+
"                           fstring = f                                        #"+
"                        strings[ fstrings.                                    #"+
"                     length-1];  int i=2; System.                             #"+
"                out.println('public class Quine{'); Syste                      #"+
"             m.out. println ('public static void main(String                  #"+
"          [] args){'); System.out. print('String t =/''); System              #"+
"        .out.print (t.replace (t.      substring(2,3),t.substring (2,3)        #"+
"        +'/'+ /n/''));t =      t.      r    epl ace('','') ; System.out.        #"+
"         println('/';');fs   t           r    ing = fstring. replace(' '       #"+
"        ,'').replace(t.su    b           s     tring (2,3),''); t=t.rep        #"+
"        lace(t.s ubstrin      g           (     2,3),'');for(int x=0;           #"+
"        x<fstring. length(    )          ;x     +=2) {try{int cs  =            #"+
"         Integer.parseInt     (        f         string. substring(x            #"+
"        ,x+1),16); int ss =    In     t       eger. parseInt (fstring.subst     #"+
"       ring( x+1,x+2),16);System.out.print(t.substring(i,i+cs).replace(         #"+
"      t. substring(0,1), '/''').replace (t.substring(1,2),'//''')); i+=         #"+
"       cs; for(in       t j = 0;j<ss; j++){System.out.print(' ');} if(ss==0)     #"+
"      {System. out.p       rintln(''); }}catch(Exception E){S       ystem.      #"+
"      out.println('exc     eption');}} System.out.println('}      }');@6       #"+
"      121811181B1711191      91A0013141A191715191B00371816       141B121       #"+
"    91710004918111617191A      00D31F1B1600ED111B1600F03       B1700F0880      #"+
"   0E71118191500E81C1A00EB1      31F1407141A181E122138       08181A1C131211     #"+
"   18181009AA1DFB06127C161AD                                  F00169A1E1880031   #"+
"3A711111715F7190012137A181BE500E                             819121CA71E00F0221A1F0";
String [] fstrings = t.split( "@");String fstring = fstrings[ fstrings. length-1];
 int i=2; System.out .println( "public class Quine{"); System.out.
   println ("public static void main(String [] args){"); System.
    out.print ("String t =\""); System. out.print (t.replace
             (t. substring(2,3), t.substring (2,3)+
              "\"+\n\""));t = t.replace(" ","");
                  System.out. println
                        ("\";");
              fstring = fstring. replace(" ","")
              .replace (t.substring (2,3),"");
              t=t.replace (t. substring(2,3), "");
for(int x=0; x<fstring. length() ;x+=2){try{int cs  =   Integer.
parseInt (fstring .substring (x,x+1),16); int ss = Integer. parseInt
(fstring.          substring( x+1,x+2),16);               System.out.
print( t.       substring(i, i+cs). replace(t.             substring(0,1),
 "\"").         replace(t. substring(1,2) ,"\\"));
i+= cs;          for(int j = 0;j<ss; j++){               System. out.print
 (" ");       }if(ss==0) {System. out.println             ("");
              }}catch( Exception E) {System.out.         println ("exception");
                 }} System.out .println("}}");
}}
```

Courtesy of Richard Bowen