Python Bonsai

This problem asks you to create two classic kinds of recursive images using the `turtle` package: a tree made of line segments and the Koch Snowflake. If you are so inclined, you can make your tree more realistic than just a stick tree and get extra credit. You can use the turtle commands to play with color, make the trunk wider, etc.

In a typical Python installation, the `turtle` drawing module is installed and ready to use.

The `turtle` package provides us with a virtual turtle that can be controlled through Python commands. Start by using the turtle package at the Python command line. To do so, load in the turtle package with the command `import turtle`. Then, play around with the turtle for a few minutes. Try commands like these:

```
>>> turtle.forward(100)    <-- turtle goes forward 100 steps

>>> turtle.right(90)       <-- turtle turns right 90 degrees

>>> turtle.penup()         <-- turtle lifts its pen up off of the paper

>>> turtle.forward(100)    <-- turtle goes forward 100 steps

>>> turtle.pendown()       <-- turtle puts its pen down on the paper

>>> turtle.pencolor("red") <-- turtle uses red pen

>>> turtle.circle(100)     <-- turtle draws circle of radius 100

>>> turtle.pencolor("blue") <-- turtle changes to blue pen (most other common
colors work too!)

>>> turtle.forward(50)     <-- turtle moves forward 50 steps

>>> turtle.xcor()          <-- turtle returns its current x-coordinate

>>> turtle.ycor()          <-- turtle returns its current y-coordinate
```

To see the complete set of `turtle` commands go to the official Python turtle page.

Note: When you run a program with turtle commands, a special window will open where the drawing will take place. This window may appear behind some of your other open windows, so you may need to move windows to see it. If you run the program again, that same window will be used again. However, when you want to finally close that drawing window, you must type `turtle.bye()` at the IDLE prompt.

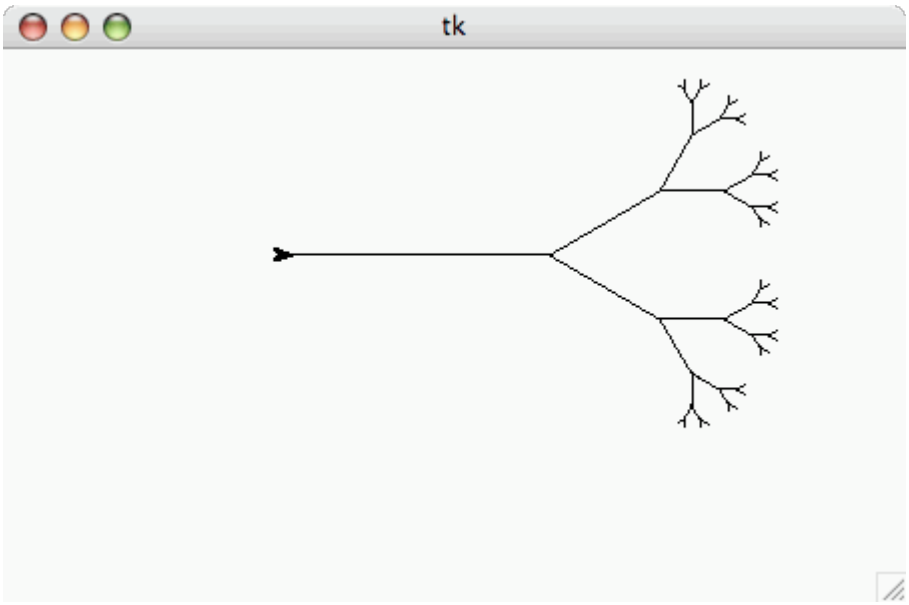
Part 1: The Stick Tree

In this problem we ask you to write a function called `svTree` that draws a stick figure of a tree. It can be sideways or upright, either way is fine. Notice that the turtle starts facing "east", so you may find it easiest to make a sideways tree.

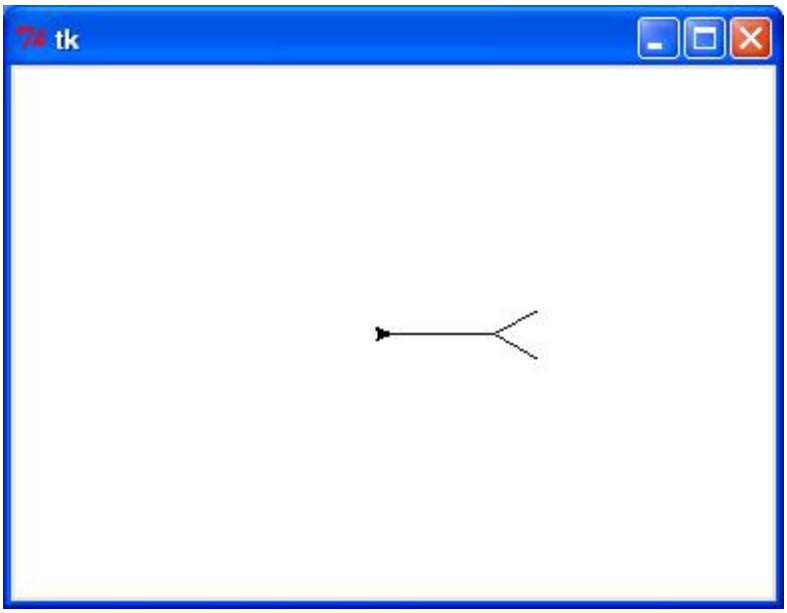
For the side view of the tree, write a function

```
svTree(trunkLength, levels)
```

Here is an example of the output from my function when `svTree(128, 6)` is run:



and another example of the output when `svTree(50, 2)` is run:



If the number of `levels` is set to 1, the tree is just a line segment (a stick)!

Note that in our program, each recursive "subtree" has a `trunkLength` half as long as it's parent's `trunkLength`. For that reason, it's convenient to run the function with `trunkLength` values that are powers of 2, since they are nicely divisible by 2. (That's why our examples above use values like 128 and 2). However, you are not obligated to make your recursive trees half the size of the parent trees - it's up to you.

If you want to make your tree face "up", you can issue a `turtle.left(90)` command before you start drawing the tree.

Important Note: You will certainly need to make recursive calls to `svTree` and you may be inclined to say `return svTree(blah, blah)` but you **don't** want to do that. If you do that, your code won't continue past this line because the `return` statement says "I'm outta here!". So, `svTree` will be called without `return`. In fact, `svTree` doesn't really need to have any `return` statements at all. When it reaches the end, it implicitly returns to where it was called from. However, for the base case you might want to draw a line and then return to whoever called you. You can do that as follows:

```
if blah blah blah:

  turtle.forward(blah)

  return   # This says:  "I'm outta here, but I'm not returning any
particular value"
```

Your tree does not need to look exactly like the one that is shown in the picture above. The branching angles can be different and the relative lengths of the "trunk" and the "branches" is also at your discretion.

The **secret to all happiness** in writing this function is the following: The turtle starts at some point. It draws the trunk of the given `trunkLength`. It turns left and draws a smaller tree (smaller `trunkLength` and `levels` reduced by 1) recursively. It then turns right and does that again. Finally, it does the appropriate combination of turning and backing up to return to the position where it started initially. You can see in the images above that the turtle has returned to its starting location (and orientation) at the end.

Why is it so important that the turtle returns to the point that it started initially? Notice that when we make our first recursive call to draw the left "subtree", we assume that when the turtle is done with that call it will be at the point where it started that recursive call. That's convenient, because it allows us to just turn a bit to the right to make the second recursive call to draw the right "subtree". Aha! So, we are expecting our "children" (recursive calls) to "behave nicely" (by returning to the point where they started), so we need to behave "nicely" too (by returning to our start position). After all, we might be someone's child too.