Tree of life

Refactoring Trees: An exercise in Research Software Engineering

In this exercise, you will convert badly written code, provided here, into better-written code.

You will do this not through simply writing better code, but by taking a refactoring approach, as discussed in the lectures.

As such, your use of `git` version control, to make a commit after each step of the refactoring, with a commit message which indicates the refactoring you took, will be critical to success.

You will also be asked to look at the performance of your code, and to make changes which improve the speed of the code.

The script as supplied has its parameters hand-coded within the code. You will be expected, in your refactoring, to make these available as command line parameters to be supplied when the code is invoked.

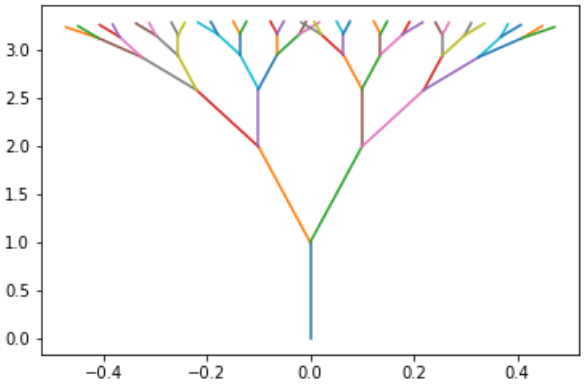
Flamel's code

Dáithí is an alchemist's assistant who has been making a name for himself by improving alchemists' working practices. One day, he receives a request from the famous alchemist Nicolas Flamel. Nicolas has also started writing code to support his work - a model to find the tree of life struccture-, but feels it can do with some improvement. He wants Dáithí to look at it and see what simple changes can be made to make the code faster and easier to understand or modify in the future, but without actually adding any new functionality.

Here's Flamel's *terrible* code:

In [1]:
```
%matplotlib inline
```

In [2]:
```
from math import sin, cos
from matplotlib import pyplot as plt
s=1
d=[[0,1,0]]
plt.plot([0,0],[0,1])
for i in range(5):
    n=[]
    for j in range(len(d)):
        n.append([d[j][0]+s*sin(d[j][2]-0.1), d[j][1]+s*cos(d[j][2]-0.1), d[j][2]-0.1])
        n.append([d[j][0]+s*sin(d[j][2]+0.1), d[j][1]+s*cos(d[j][2]+0.1), d[j][2]+0.1])
        plt.plot([d[j][0], n[-2][0]],[d[j][1], n[-2][1]])
        plt.plot([d[j][0], n[-1][0]],[d[j][1], n[-1][1]])
    d=n
    s*=0.6
plt.savefig('tree.png')
```



Rubric and marks scheme

Part one: Refactoring (15 marks)

- Copy the code above into a file `tree.py`, invoke it with `python tree.py`, and verify it creates an image `tree.png` which looks like that above.
- **[1 mark]** Initialise your git repository with the raw state of the code.
- **[10 marks]** Identify a number of simple refactorings which can be used to improve the code, *reducing repetition* and *improving readability*. Implement these one by one, with a git commit each time.
  - 1 mark for each refactoring, 1 mark for each git commit, at least five such.
- Do NOT introduce NumPy or other performance improvements yet (see below.)

- **[4 marks]** Identify which variables in the code would, more sensibly, be able to be input parameters, and use Argparse to manage these.
  - 1 mark for each of the arguments identified.

Part two: Performance programming (10 marks)

- **[5 marks]** For the code as refactored, prepare a figure which plots the time to produce the tree versus the number of iteration steps completed. Your code to produce this figure should run as a script, which you should call `perf_plot.py`, invoking a function imported from `tree.py`. The script should produce a figure called `perf_plot.png`. Comment on your findings in a text file, called `comments.md`. You should turn off the actual plotting, and run only the mathematical calculation, for your performance measurements (add an appropriate flag to allow that). 1 mark awarded for each of the following:
  - Time to run code identified.
  - Figure created.
  - Figure correctly formatted.
  - Figure auto-generated from script.
  - Performance law identified.

- **[5 marks]** The code above makes use of `append()` which is not appropriate for NumPy. Create a new solution (in a file called `tree_np.py`) which makes use of NumPy. Compare the performance (again, excluding the plotting from your measurements), and discuss in `comments.md`. 1 mark awarded for each of the following:
  - NumPy solution uses array-operations to subtract the change angle from all angles in a single minus sign.
  - Taking the sine of all angles using `np.sin`.
  - Moving on all the positions with a single vector displacement addition.
  - Numpy solution uses `hstack` or similar to create new arrays with twice the length, by composing the left-turned array with the right-turned array.
  - Performance comparison recorded.

As with assignment one, to facilitate semi-automated marking, submit your code to moodle as a single Zip file (not .tgz, nor any other zip format), which unzips to produce files in a folder named with your **student number**.