DM 536 Introduction to Programming

Fall 2013 Project (Part 1)

Department of Mathematics and Computer Science
University of Southern Denmark

September 15, 2013

Introduction

The purpose of the project for DM536 is to try in practice the use of programming techniques and knowledge about the programming language Python on small but interesting examples.

There are two possible projects. You have to pick one of these. Each project consists of two parts. You may choose to do the first part of one project and the second part of the other project.

Please make sure to read this entire note before starting your work on this part of the project. Pay close attention to the sections on deadlines, deliverables, and exam rules.

Exam Rules

This first part of the project is a part of the final exam. Both parts of the project have to be passed to pass the overall project.

Thus, the project must be done individually, and no cooperation is allowed beyond what is explicitly stated in this document.

Deliverables

A short project report (at least 4 pages without appendix) has to be delivered. This report has to contain the following 7 sections:

- **front page** (course number, name, section, date of birth)
- **specification** (what the program is supposed to do)
- **design** (how the program was planned)
- **implementation** (how the program was written)
- **testing** (what tests you performed)
- **conclusion** (how satisfying the result is)
- **appendix** (complete source code)

The report has to be delivered as a single PDF file electronically using Blackboard's SDU Assignment functionality. Do not forget to include the source code int the appendix!

Deadline

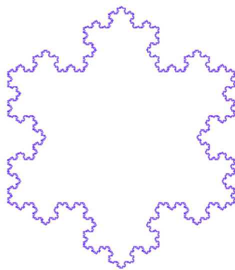
Friday, October 4, 23:59

Project “Fractals and the Beauty of Nature”

Fractals are geometric objects that are similar to themselves on arbitrarily small scales. There are many examples of fractals in nature, and they form some of the most beautiful structures you can find. One example is snowflakes, but also lightning has a fractal structure. Another example are ferns, where each part is similar to the whole.

In this project, we will use Swampy to generate fractals that are similar to structures found in nature.

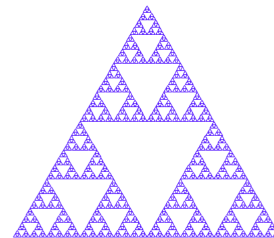
Task 0 – Preparation



On the course home page you find an example using a Koch curve to render a snowflake (`koch.py`, `FractalWorld.py`, `koch2.png`, and `koch5.png`). Use this example and experiment with the depth and the length parameters. Make sure you use the updated file `FractalWorld.py`. Understand what (new) features are used here. In particular, line width and color options have been added to `fd`.

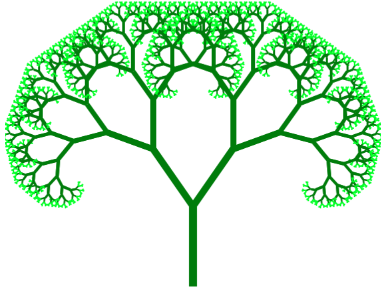
Task 1 – Sierpinski Triangle

A Sierpinski triangle is a triangle divided into three triangles, that are demselves divided into three triangles etc. On the course home page, you find four examples (`sierpinski0.png`, `sierpinski1.png`, `sierpinski2.png`, and `sierpinski5.png`). On the right-hand side you see a Sierpinski triangle of depth 5, i.e., a triangle that has been subdivided five times.



You task is to write a Python program that draws a Sierpinski triangle. To this end, you can follow the approach of the Koch Snowflake, i.e., at depth 0 you draw a triangle. At any other depth n, you draw Sierpinski triangle of depth $n-1$, move to the next corner, draw another Sierpinski triangle of depth $n-1$, move to the last corner, draw the third Sierpinski triangle of depth $n-1$, and, finally, move to the original corner.

Task 2 – Binary Tree

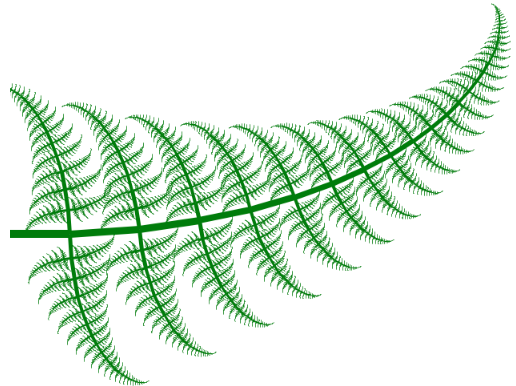


A binary tree is a tree, which is obtained by first drawing a stem and then subdividing into two branches. A tree consisting of just a stem is called a tree of depth 1. The picture to the left shows a tree of depth 12. On the course home page, you find examples (`tree3.png`, `tree6.png`, `tree9.png`, and `tree12.png`). Here, as an addition, the last two layers of branches have been colored `darkgreen` instead of `green`.

You task is to write a Python program that draws a binary tree. To this end, you have to modify the approach from the previous tasks slightly. The main idea is to draw a line, then turn and draw the left branch, then turn and draw the right branch, and, finally, go back where you came from.

Task 3* – Fern Time

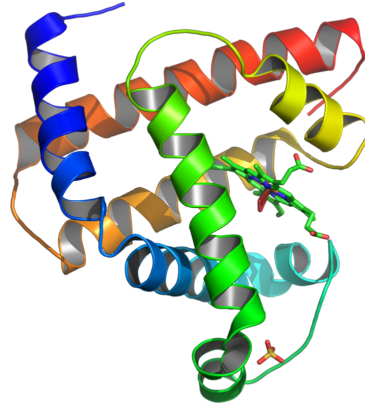
There are three differences between a fern and a binary tree. First, the fern has three branches from each stem and each subbranch. Second, the fern use some small constant angle to turn the middle branch. Third, the middle branch is scaled down less than the left and the right branch. On the course home page, you find examples (`fern03.png`, `fern06.png`, `fern12.png`, and `fern24.png`).



Your challenge task is to write a Python program that draws a fern. To this end, you have to modify the approach from the previous task. The main idea is that due to the third difference, a constant recursion depth is not useful in implementing a fern. Instead, define a limit for the size of the subbranches drawn and use this as the base case of the recursion.

Note that this task is optional and does not have to be solved for this part of the project to be considered as passed.

Project “From DNA to Proteins”



In nature, deoxyribonucleic acid (short: DNA) is used to encode genetic information of living organisms as sequences of bases. There are four bases found in DNA: adenine (short: `A`), cytosine (short: `C`), guanine (short: `G`), and thymine (short: `T`). One of the main functions of DNA is to encode the sequence of amino acids used in the construction of proteins.

Modern technological advances have made it possible to decipher this genetic information. You can find for example the base sequences of human chromosomes on the following web site:

`http://hgdownload.cse.ucsc.edu/downloads.html`

In this project, we will assemble base sequences from such files and analyze these sequences to identify proteins.

Task 0 – Preparation

Download the file `chromFa.tar.gz` from the full data set for human and unpack it. Locate the files `chr1_gl000191_random.fa` and `chrX.fa` and view them in a text viewer or editor.

The first line is a short description of the sequence contained in the file. The rest of the lines is the base sequence. You will find lower-case and upper-case variants of the bases `A`, `C`, `G`, and `T`. In addition you will find `N`, which for the purpose of this project can be ignored.

Task 1 – Assembling the Sequence

For our purposes, we will ignore `N`s and we will ignore the difference between lower-case `a`, `c`, `g`, `t` and upper-case `A`, `C`, `G`, `T`.

Your task is to write a program that reads all lines and constructs one long string containing the base sequence. In this process, whitespace and `N`s have to be ignored. In addition, all base pairs should be represented by upper-case letters. Use the smaller file to print the result and compare the beginning of the assembled sequence manually to the original file.

Task 2 – Finding Starting Points

The construction of a protein by a ribosome begins at a start codon. On our DNA, this is denoted by the base sequence `ATG`. This is called a *start codon*. Approximately 25 bases earlier in the sequence we often find a so-called TATA box. This is a base sequence `TATAAA`.

Your task is to write a function that will find all positions of a start codon that occurs 15-30 bases after the beginning of a TATA box. To this end, first write a function that will find all positions of the string `TATAAA` in the sequence. Then, find out how far the next ATG is located. If the distance is inside the admissible range, remember the index of the start codon.

Task 3 – Finding End Points

Each protein is encoded by a sequence of bases starting with ATG. Every amino acid is encoded by three bases. The end of a protein is given by a *stop codon*. This can be any of the following three sequences: `TAG`, `TAA`, or `TGA`.

Your task is to write a function that will identify the end point of a protein, i.e., the index of the first stop codon encountered after the start codon. To this end, you need to advance in steps of 3 through the base sequence until you encounter either the end of the sequence or a stop codon.

Task 4* – Potential Proteins without TATA Boxes

Not all proteins are prefixed by a TATA box occurring 15-30 bases earlier. In general, potential proteins can be overlapping. For example, the base sequence `ATGAATGAATAGATGA` contains a potential protein at index 0 (`ATGAATGAATAG`) and at index 4 (`ATGAATAGATGA`). We call this a genuine overlap. There is also trivial overlap, when `ATG` occurs inside a base sequence at a position divisible by three, e.g., `ATGATGTAG`.

Your challenge task is to identify all potential starts of proteins and answer the following two questions w.r.t. the file `chrX.fa`:

- How many genuine overlaps are there?
- How many potential proteins are there? Here, for genuine overlaps both proteins count while for trivial overlaps, only the longest protein counts.

Note that this task is optional and does not have to be solved for this part of the project to be considered as passed.