DM550/DM857
Introduction to Programming
Fall 2017 Project (Python)

Department of Mathematics and Computer Science
University of Southern Denmark

October 13, 2017

Introduction
The purpose of this project is to try in practice the use of general programming techniques as well as knowledge about the programming language Python. Please make sure to read this entire note before starting your work on this project. Pay close attention to the three sections below.

Exam Rules
This project is part of the final exam for DM550 and DM857. Both the project qualification assessments, this Python project, and the Java project have to be passed in order to pass the overall exam. The project should be done in groups of 3 students. Other group sizes require written confirmation.

Deliverables
A project report (10-20 pages without appendix) containing the following 6 sections has to be delivered:

- **front page** (course number, names, sections, dates 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)

The report has to be delivered as a PDF file (suffix .pdf) together with the Python source code files (suffix .py) and any needed supporting data files electronically using Blackboard's SDU Assignment functionality. In addition, one printed copy has to be delivered to the "dueslag" of the respective teaching assistant. **Do not forget to include the complete source code!**

Deadlines
The deadline for this project is

Friday, October 27, 23:59.

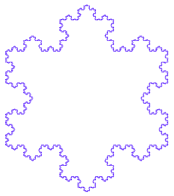
Late deliveries cannot be accepted, so please plan your time accordingly.

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.

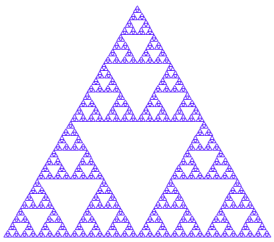
In this project, we will use the `turtle` module to generate fractals that are similar to structures found in nature.

Task 1 – Preparation I



On the course home page you find an example using a Koch curve to render a snowflake (`koch.py`), with the expected results shown in `koch.png`. Use this example and experiment with the depth and the length parameters. Consult the documentation on the `turtle` module to understand the features used here.

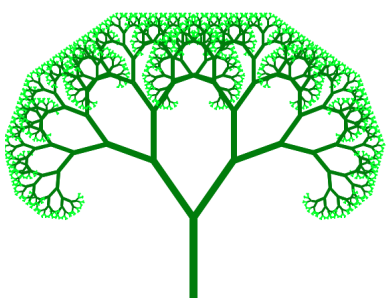
Task 2 – Sierpinski Triangle



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 3 – Binary Tree

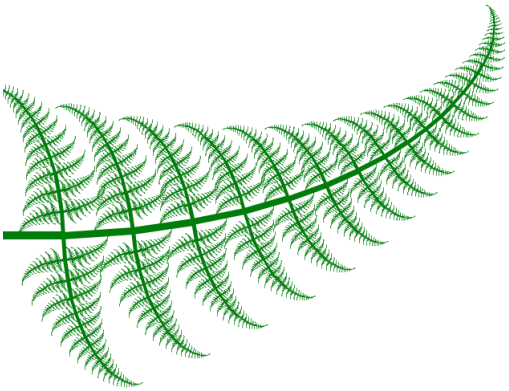


A binary tree is a tree, which is obtained by first drawing a stem and then subdividing into two branches. A tree consisting of just a stem is called a tree of depth 1. The

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 4 – 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 uses 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 (`fern10.png`, `fern3.png`, `fern1.png`, and `fern03.png`).

Your task is to write a Python program that draws a fern. To this end, you can either develop a new approach or modify the approach from the previous task.

Hint: You might find it beneficial to rely on a limit for the size of the subbranches drawn rather than a constant recursion depth. Instead, you might define a limit for the size of the subbranches drawn and use this as the base case of the recursion. Note that this modification is an optimisation, not a requirement.

Fractal Description Language

Many fractals can be generated from descriptions in the Fractal Description Language (.fdl). This language describes a start state of a fractal and a set of rules how to progress to larger depths.

For example, to generate a Koch curve, we start with the initial (depth 0) state `F` signifying a forward move, i.e., a straight line. The rule for expanding to the next depth is given as a replacement rule.

```
F -> F L F R F L F
```

where `L` is a 60 degree turn to the left and `R` is a 120 degree turn to the right. That is, we replace a straight line by a straight line, a left-turn, a straight line, a sharp right-turn, a straight line, a left-turn, and a fourth and final straight line.

Thus, the state for depth 1 is `F L F R F L F`. To get to depth 2, we have to apply the rule again to all positions of the state where it is possible, i.e., we have to replace each of the four `F` by `F L F R F L F`. The result is F L F R F L F L F L F R F L F R F L F L F L F R F L F R F L F L F L F R F L F where the new sections are underlined to aid your understanding.

To get to depth 3, we would have to replace each of the 16 `F`s by the right side of the rule. For the sake of brevity, I leave this exercise to you.

Let us take a look at the file `koch.fdl` available from the project section of the course home page.

```
start F
rule F -> F L F R F L F
length 2
depth 5
cmd F fd
cmd L lt 60
cmd R rt 120
```

The first line give the start state, i.e., the state `F` for depth 0. The second line give the only rule needed for the Koch curve, i.e., to replace `F` by `F L F R F L F`. The third line specifies the length of each segment, i.e., each straight line will be 2 units long. The fourth line specifies that states should be expanded to depth 5 before drawing the fractal. Finally, the Lines 5 to 7 specify that `F` is a straight line, `L` is a 60 degree left-turn and `R` is a 120 degree right-turn.

Task 5 – Preparation II
Download all the .fdl files from the course homepage (at least `dragon.fdl`, `fern.fdl`, `koch.fdl`, `sierpinski.fdl`, `sierpinski2.fdl`, `snowflake.fdl`, and `tree.fdl`).

Try to understand how the rules generate their respective fractals. While most of the fractals are known from earlier tasks, `dragon.fdl` represents a dragon curve and `sierpinski.fdl` represents a Sierpinski curve.

Task 6 – Applying One Rule
A rule consists of a single letter for the left side and a list of letters for the right side. Your task is to create a function `apply(left,right,state)` that applies the rule to each matching element of the list. For a function call `apply("F",["F","L","F","R","F","L","F"],["F","R","F","R","F"])` it should generate the list `[["F","L","F","R","F","L","F"],"R",["F","L","F","R","F","L","F"],"R",["F","L","F","R","F","L","F"]]`.

Task 7 – Flattening After applying one or more rules, the list representing the fractal is a mixed list consisting of both letters and lists as elements. Write a function `flatten(state)` that turns it into a simple list of letter. For a function call `flatten([["F","L","F","R","F","L","F"],"R",["F","L","F","R","F","L","F"],"R",["F","L","F","R","F","L","F"]]` the expected result is `["F","L","F","R","F","L","F","R","F","L","F","R","F","L","F","R","F","L","F","R","F","L","F"]`.

Task 8 – Increasing Depth Write a function `step(rules,state)` that takes a list `state`, applies all rules in `rules`, and finally flattens the resulting list.

Then write a function `compute(depth,rules,start)` that uses the `step` function `depth` times to transform the list `start` into a list representing a fractal of depth `depth`.

Task 9 – Representing Rules (supplementary) Define a user-defined type (=Python class) to represent a rule. Use methods instead of functions for the functionality from Tasks 6–8, whenever this makes sense.

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

Task 10 – Preparation III
Try to understand the other parts of the fdl files, in particular the commands.

Task 11 – Executing Commands
A command consists of a command string (such as `fd`, `bk`, `lt`, `rt`, `scale`, `nop`) and a list of arguments. Your task is write a function `execute(turtle,length, cmd,args)` for executing a command with its arguments and returning the current length. This function gets as the first two arguments a turtle object and a length.

The command with command string `lt` and argument list `["60"]` should execute the Python statement `turtle.lt(60)` and return `length`. The command with command string `fd` should execute the Python statement `turtle.fd(length)` and return `length`. The command `scale` should multiply `length` with the given float and return it. Finally, the command `nop` simply does nothing (no operation) and returns `length`.

Task 12 – Loading a Fractal Description Language File
A fractal is described by a start state, a list of rules, a mapping from single letters to commands, a length, and a target depth.

Your task is to create a function `parse(fdl)` that takes as argument the name of an fdl file and reads the different elements from the fdl file into appropriate data structures.

Task 13 – Loading a Fractal Description Language File
Your final task is to create a function that `run(turtle,fdl)` that takes as argument a turtle and the name of an fdl file. The function first uses `parse` to read the fdl file. Then it uses `compute` to compute the list of letters representing the fractal at the given target depth. Finally, it uses `execute` to execute all the commands represented by this list of letters.

Task 14 – Support for Pen Size and Colors (supplementary)
The fractals look nice enough, but some colors and wider lines would make them more pretty. Your supplementary task is to extend the Fractal Descripton language by the commands `color` and `width` where color gets a color name, a color code or "`random`" as an argument while width gets a float. Random colors can e.g. be generated by using the `randint` function from the `random` module and format strings:
`"#%02x%02x%02x" % (randint(0,255),randint(0,255),randint(0,255))`

Here is an example for a nicer dragon curve:

```
start F X
rule X -> X R Y F
rule Y -> F X L Y
length 3
depth 13
color random
width 2.0
cmd F fd
cmd X nop
cmd Y nop
cmd L lt 90
cmd R rt 90
```

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

Task 15 – Taking it to the third dimension (challenge) The fractals described by the Fractal Description Language are represented as Lindenmayer-systems or short L-systems. Such systems can be extended to the third dimension using rotational matrices. This extension is described in Chapter 1, Section 1.5 of the book "The Algorithmic Beauty of Plants" (`http://algorithmicbotany.org/papers/abop/abop.pdf`).

Implement this extension and use a Python-based 3D visualization system such as GlowScript or Panda3D to visualize 3D fractals such as the space-filling Moore curve.

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

Task 16 – Making it real (challenge) Bracketing and randomization are two techniques to create more realistic fractals. They are also9 described in the book "The Algorithmic Beauty of Plants" (`http://algorithmicbotany.org/papers/abop/abop.pdf`).

Implement these extensions, either for the 2-dimensional or 3-dimensional L-systems. Then build trees, bushes, flowers, or completely different structures that can be used as background in a computer animation or computer game. Animate the objects to move, e.g. according to wind.

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