DM536 / DM550 Part I
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM536/

PROJECT PART 2

Organizational Details

- 2 possible projects

- projects must be done individually, so no co-operation
- you may talk about the problem and ideas how to solve them

- deliverables:
  - Written 6 page report as specified in project description
  - handed in electronically as a PDF + source code files
  - ~~pre-delivery deadline: October 2, 23:59~~
  - FINAL deadline: October 23, 23:59

- ENOUGH - now for the FUN part …

Fractals and the Beauty of Nature

- geometric objects similar to themselves at different scales

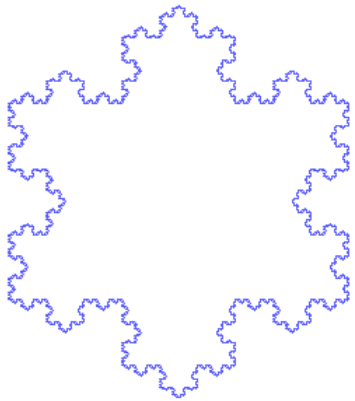
- many structures in nature are fractals:
  - snowflakes
  - lightning
  - ferns

- **Goal:** generate fractals from Fractal Description Language

- **Challenges:** Representation, Interpretation, File Handling

Fractals and the Beauty of Nature

- Task 4: Preparation II
  - understanding descriptions given in .fdl files



```
F fd
L lt 60
R rt 120
```

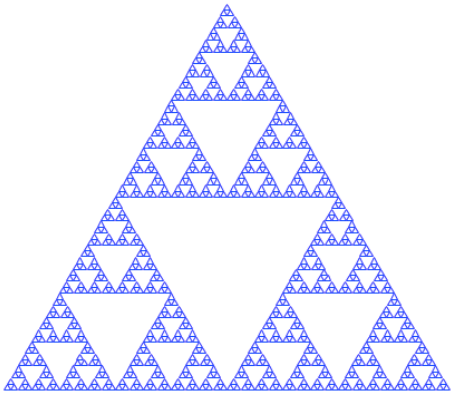
- Task 5: Rules
  - representing and applying rewriting rules

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

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

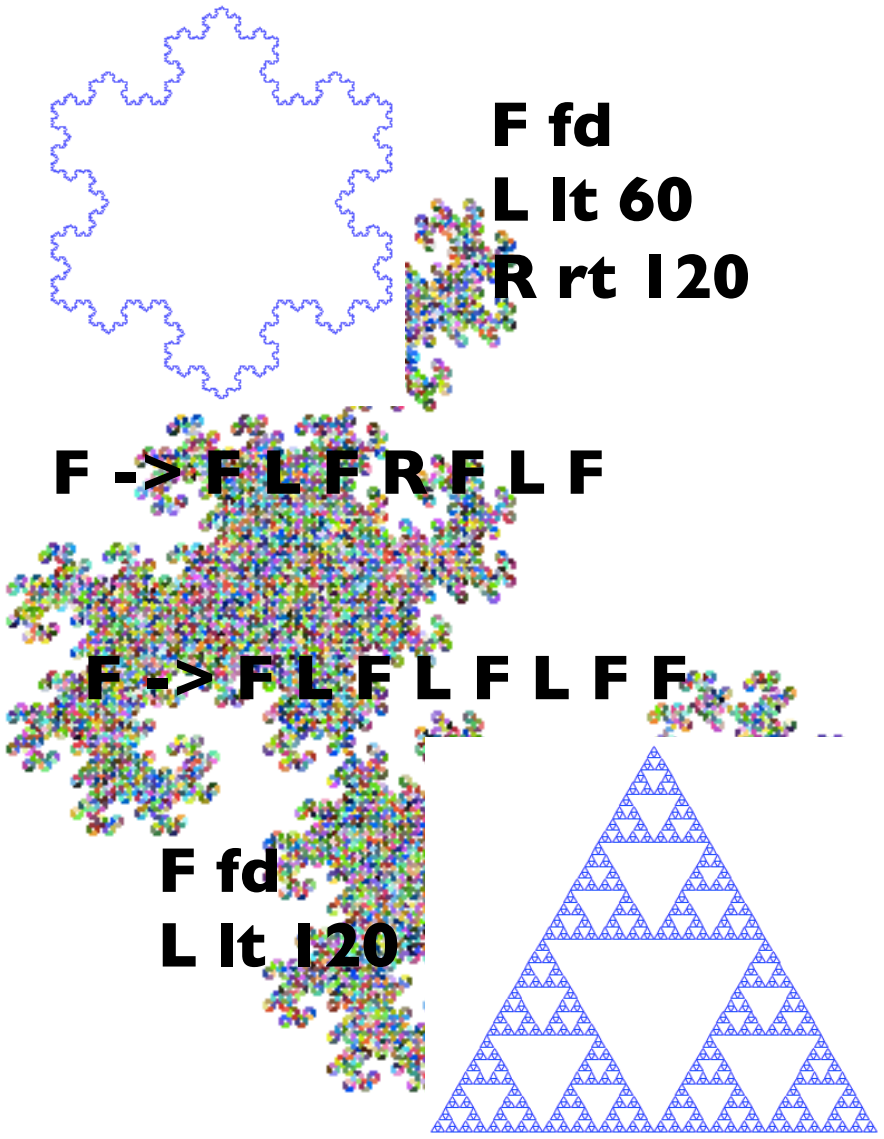
- Task 6: Commands
  - representing and executing turtle commands

```
F fd
L lt 120
```



Fractals and the Beauty of Nature

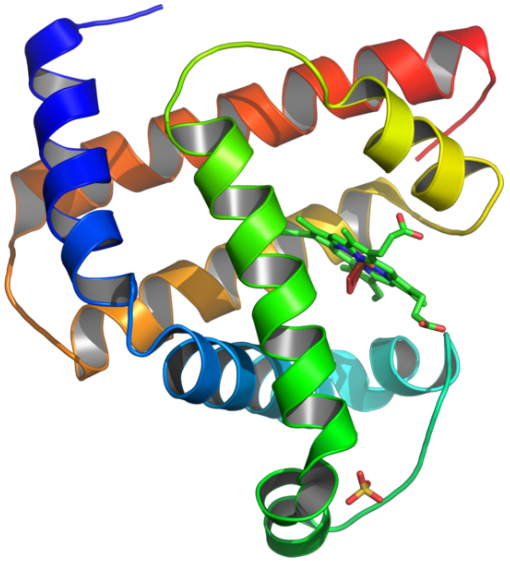
- Task 7: Loading Files
  - load and interpret fractal descripton language files
- Task 8: Generating Fractals
  - compute new states and draw the fractal
- Task 9 (optional): Colors / LW
  - add support for colors and line widths



From DNA to Proteins

- proteins encoded by DNA base sequence using A, C, G, and T

- Background:
  - proteins are sequences of amino acids
  - amino acids encoded using three bases
  - chromosomes given as base sequences



- **Goal:** build proteins from base sequences

- **Challenges:** Nested Data Structures, Representation

From DNA to Proteins

- Task 5: Preparation II
  - output base sequences
- Task 6: Representing Amino Acids
  - create user-defined type and read instances from file
- Task 7: Setting up the Translation
  - create user-defined type Ribosome as translator
- Task 8: Creating Proteins
  - represent and assemble proteins as amino acid sequences
- Task 9 (optional): Representing Codons
  - replace strings of length 3 by a user-defined type

SELECTING DATA STRUCTURES

Reading and Cleaning Words

1. read file given as argument
2. break lines into words
3. strip whitespace & punctuation
4. convert to lower-case letters

- import module sys for command line arguments sys.argv
- Example: `import sys; print sys.argv`

- import module string for punctuation
- Example: `import string; print string.punctuation`

- use `translate(None, deletechars)` to remove punctuation
- Example: `"Hello World!".translate(None, "ol")`

Word Frequency in E-Books

1. use program on Project Gutenberg e-book
2. skip over beginning & end of ebook (marked "***")
3. count total number of words
4. count number of times each word is used
5. print 20 most frequently used words

- use Boolean flag to indicate when to start
- use list to gather all words (and count total number)
- use dictionary to count number of times each word is used
- use tuple comparison to sort words

Optional Parameters

- have seen functions that take variable length argument list

- also possible to make some parameters optional
- in this case, default value has to be supplied by programmer
- Example:

```
def print_most_common(hist, num = 10):
    t = most_common(hist)
    print "The most common", num, "words are:"
    for n, word in t[:num]:
        print word, "\t", n
print_most_common(freq, 20)
```

Dictionary Subtraction

1. find all words that do NOT occur in other word list

- to this end, subtract dictionaries from each other
- **Idea:** new dictionary containing with keys only in first dict
- Implementation:

```
def subtract(d1, d2):
    d = {}
    for key in d1:
        if key not in d2:
            d[key] = None
    return d
```

Random Number Generation

- to work with random numbers, import module random
- Example: import random

- function random() returns random float from 0.0 to < 1.0
- Example: for i in range(10): print random.random()

- function randint(a, b) returns random integer in range(a,b+1)
- Example: for i in range(10): print random.randint(1,10)

- function choice(seq) returns random element of a sequence
- Example: random.choice("Slartibartfast")
  random.choice([23, 42, -3.0])

Random Words

1. choose random word from histogram according to frequency

- how to ensure random choice w.r.t. frequency?
- **Idea 1:** create list with n copies of word with frequency n
- Implementation:

```
def random_word(h):
    t = []
    for word, n in h.items():
        t.extend([word] * n)
    return random.choice(t)
```

- works, but very inefficient!

Random Words

- **Idea 2:** use list with cumulative sum of frequencies
- Implementation:

```
def random_word(h):
    words = h.keys();  sum = 0;  cum = []
    for word in words:  sum += h[word];  cum.append(sum)
    num = random.randint(1, cum[-1]); low = 0; high = len(cum)-1
    while low < high:
        mid = (low+high) / 2
        if num <= cum[mid]:  high = mid
        elif num > cum[mid]:  low = mid+1
    return words[low]
```

Markov Analysis

1. generate more meaningful random texts

- word order in texts is not random
- markov analysis maps a finite number of words (prefix) to all possible following words (suffix)

- how to represent the prefixes?

- how to represent the collection of possible suffixes?

- how to represent the mapping from prefixes to suffixes?

Data Structures

- for mapping, we clearly use a dictionary
- for prefixes, we need to be able to "shift" them (list?)
- we also need to use them as dictionary keys
- thus, we use tuples to present prefixes (+ slicing and "*")
- for suffixes, we need to add elements (list? dictionary?)
- we also need to efficiently generate random word (list?)
- tradeoff space vs time
  - dictionary uses less space and easy to add
  - list uses less time for generating a word
  - can change representation before generation

Debugging Hard Bugs

- bugs can be hard to find

- four popular strategies
  1. reading: re-read your code, check that it is right!
  2. running: make changes, experiment with outcome
  3. ruminating: take time to think it over (and over)
  4. retreating: revert to a known-to-be-good version

- often combination of these strategies needed
- always good to view debugging as scientific experiment

FILE HANDLING

Persistence

- persistent = keeping (some) data stored during runs
- transient = beginning from input data each time over

- most programs so far have been transient

- examples of persistent programs:
  - operating systems
  - web servers
  - most app(lication)s on recent Android, iOS, and Mac OS X

- text files are easiest way to save some program state
- alternatively, program states can be saved in databases

Writing to a File

- we know how to read a file using open(name)
- we can specify read/write mode using open(name, mode)
- Example:
```
f1 = open("anna_karenina.txt", "r")
f2 = open("myfile.txt", "w")
```

- use method write(str) of file object to append string to file
- Example:
```
f2.write("This is my first line!\n")
f2.write("This is my second line!\n")
```
- each invocation of write(str) will append, not overwrite!

- when you are finished with a file, please close() it
- Example:
```
f1.close()
f2.close()
```

Format Operator

- values need to be converted to a string for use in write(str)
- for single value, the str(object) function can be used
- Example: f.write(str(42))

- alternatively, use *format operator* "%"
- Example: f.write("%d" % 42)
  f.write("The answer is %d, my friend!" % 42)
- first argument *format string*, second argument value
- format sequence %d for integer, %g for float, %s for string

- for multiple values, use tuple as value
- Example: f.write("The %s is %g!" % ("answer", 42.0))

Directories

- file are organized in *directories*
- every program has a *current directory*
- the current directory is used by default, e.g. for open(name)
- get current directory by importing getcwd() from os module
- Example:
```
import os
print os.getcwd()
```
- change current working directory by using chdir(path)
- Example:
```
os.chdir("..")
print os.getcwd()
```
- list contents of a given directory by using os.listdir(path)
- Example:
```
print os.listdir("dm502")
```

Filenames and Paths

- path = directory & file name
- *relative paths* start from current directory
- Example:

```
path1 = "dm536/tools/anna_karenina.txt"
```

- *absolute paths* are independent from current directory
- Example:

```
path2 = "/Users/petersk/sdu/dm536/tools/anna_karenina.py"
```

- can be obtained from relative path using os.path.abspath(path)
- Example:

```
path3 = os.path.abspath(path1)
```

Operations on Paths

- check whether a directory or file exists using os.path.exists
- Example: os.path.exists(path1) == True
  os.path.exists("no_name") == False

- check whether a path is a directory using os.path.isdir
- Example: os.path.isdir(path1) == False
  os.path.isdir("..") == True

- check whether a path is a file using os.path.isfile
- Example: os.path.isfile(path1) == True
  os.path.isfile("..") == False