Università della Svizzera italiana

Faculty of Informatics

Programming Fundamentals I
Fall 2014
Topic 6: Recursion and Text Files

Prof. Mehdi Jazayeri

Assisted by: Anton Fedosov, Yuriy Tymchuk, Rui Xin

Week 6: 27 October - 2 November, Due: 3 November

Instructions

Please read these instructions carefully.

Each week in the first ten weeks of the semester, you are supposed to master a topic and demonstrate your mastery to the teaching staff. To gain mastery, you should attend the lecture, study the topic, and do many exercises on the week's topic. The more exercises you do, the more confidence you will gain in your ability and your understanding. We will post some questions on iCorsi that you can use to test yourself. Also, the textbook has exercises that you can use to test yourself. Make sure you can answer ALL the exercises in the book.

To show your mastery of the topic, you will typically be given a program to study and then modify it to enhance its features. We will also give you a list of topics you could explore on your own if you want to extend your knowledge.

Finally, in the Monday atelier session, we will ask you to write a program and explain how it works.

For any questions, feel free to post on the iCorsi questions forum for the course or ask during the atelier session.

Week 6

Topic: This week you will work with lists, files and recursive functions.

Introductory questions:

1. Write a function that returns the number of blank lines in a textfile.

2. Write a function that reads the lines from a textfile and writes them to a newly created textfile.
3. Write a function that reads the lines in a textfile and returns the average line length in the file.
4. Write a function that uses recursion to print the elements of its input list.
5. Write a function that takes two lists as parameters and checks whether the first is a sublist of the second. Note that the elements in the lists may be lists themselves.
6. Write a function that "flattens" its input list.
7. Write a function that implements the slice operation for lists.
8. Write a function that accepts a list of strings and capitalizes all the elements of the list.
9. Write a function that accepts a list that contains nested lists and returns the "depth" of the list.
10. Write a function that implements equality testing for lists, taking two lists and returning a Boolean.
11. Write a function that implements the reverse operation for lists.
12. Write a function that accepts an integer and a list of integers and *partitions* the list such that all elements smaller than the integer will occur before all the elements that are greater than the integer. Do not sort the list.
13. Write a function that accepts two strings and checks if they are anagrams of one another.
14. Write a function that returns the reverse of its input string. Use the function to write another function to test if its input string is a palindrome.
15. Write a recursive function to test if its input string is a palindrome.

Program to modify: Koch curve

Fractal[1] is an interesting mathematical concept, which basically refers to a figure that repeats a specific pattern in some way. As a result, a fratcal can usually be observed to be similar to part of itself.

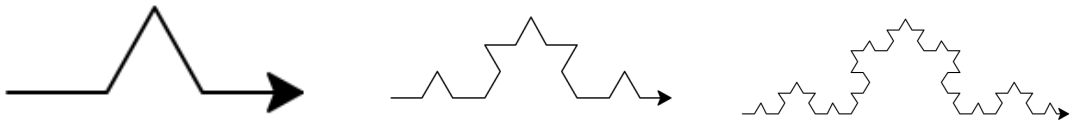
In fact, such repetition is related to recursion in programming. In **Exercise 5.4** of *Think in Python*, there is an example of fractal, **Koch curve**. We provide the solution of this problem below, which can draw the Koch curve as Figure 1 shows.

```
import turtle

turtle.speed(0)
def koch(step):
    if step < 30:
        turtle.fd(step)
        return
    koch(step/3)
    turtle.lt(60)
    koch(step/3)
    turtle.rt(120)
    koch(step/3)
    turtle.lt(60)
    koch(step/3)
```

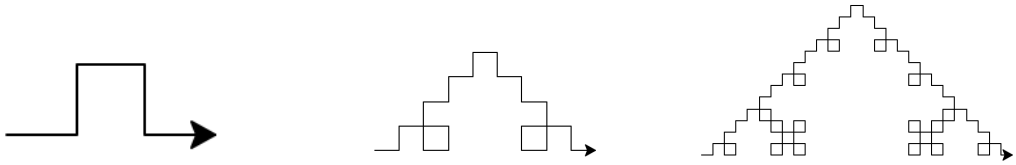
Please read the code carefully, make sure you understand every step of it, and modify the code to draw the square Koch curve as Figure 2 shows.

[1] http://en.wikipedia.org/wiki/Fractal



(a) step=80 (b) step=200 (c) step=300

Figure 1: Koch curve



(a) step=80 (b) step=200 (c) step=300

Figure 2: square Koch curve

Program to write:

In a text processing software like **Apple Pages** or **Microsoft Word** it is possible to inquire word statistics in the current document. Often one may have a limit on the number of words that one may include in a paragraph or a page. Write a program that reads a file *text.txt* (provided). Your program should calculate and print the following statistics about the text file:

- Number of lines in a file
- Number of words in a file
- Number of characters in a file excluding spaces

To study further: During a program's execution it maintains some state information such as values, variables that point to them, and functions that are being executed. This state is usually referred as Call Stack.[2]. You can see the visualisation of this stack for the simple example here: `http://goo.gl/uEuuqB` by using "Forward" and "Back" buttons below the code snippet.

Now consider a recursive implementation of Fibonacci number calculation:

```
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fib(n - 1) + fib(n - 2)
```

You can analyse the call stack of this function here: `http://goo.gl/2vIujs`. As you can see,

[2] `http://en.wikipedia.org/wiki/Call_stack`

this execution generates a "large" stack, and it will grow even larger with big input values. This may result in memory overflow and program crashes, so you have to keep in mind this execution property of recursive programs.
For comparison let's take this iterative approach for Fibonacci number calculation:

```
def fib(n):
  a = 0
  b = 1
  for _ in range(n):
    old_b = b
    b = a + b
    a = old_b
  return a
```

Interactive example of the execution of this code can be found here: `http://goo.gl/Q5ipIi`. You can notice that the execution does not generate such a big stack trace.
We encourage you to use the Python Tutor[3] visualisation tool, to analyse call stacks of other programs.

[3] `http://www.pythontutor.com/visualize.html`