Homework Set #1

Imperative Progamming with Python

January 2012

The homework should be uploaded using the BlackBoard system, it should *not* be printed. You should upload a compressed (`.zip`) file containing:

- A text (`txt`, Word, `ps`, PDF) file with all the answers to the exercises that do not contain source code (in the answer).
- For each exercise including source code, a directory named '`exerciseN`' where `N` is the number of the exercise. This directory should include *all* the needed files and directory structure to run the program (if applicable).
- When an exercise asks you to modify or write a variant of a previous program you should include all the versions, not just the last one.

Homework not uploaded by the due date will be considered failed. The overall score is 100/100. You may choose to do the exercises you want, but keep in mind that those marked with a star (★) are *mandatory*.

For every exercise that you do, estimate the amount of time that it will take you and write it down (e.g., 5–10 mins). After doing it, write the real amount of time that you spent. You will not be penalized if they don't match but you *will* be penalized if you don't do it. Include this data in the answer of the exercise. **Due date:** 16/01/2012.

Exercise 1. (5 pts.) Give a short (at most few paragraphs long) comparison of the Functional, Imperative and Declarative programming paradigms.

Exercise 2. (5 pts.)

a) Find out exactly which positive number is the boundary between Integers and Longs.

b) Which method did you use to find it? Random trial and error? Wikipedia? Any algorithm?

Exercise 3. (5 pts.) Explain why the logical connectives behave like this when used on strings:

```
>>> 'skate' or 'die'
'skate'
>>> not 'die'
False
```

In which case would this be useful? You can check the Python documentation to solve the exercises.

Exercise 4. (5 pts. ★) Explain the difference (in behaviour) between the following two programs

```
if A:
    f1()
elif B:
    f2()
elif C:
    f3()
```

```
if A:
    f1()
if B:
    f2()
if C:
    f3()
```

Exercise 5. (5 pts. ★) Write a function `absolute(n)` which, given a number n, returns its absolute value. You should not use the built-in function `abs`.

Exercise 6. (5 pts.) Do exercise 5 from Chapter 3 in the book: "Drawing a grid". Remember to use functions.

Exercise 7. (5 pts.) Do exercise 1 from Chapter 5 in the book: "Fermat's last theorem".

Exercise 8. (10 pts.) Do exercise 6 from Chapter 6 in the book: "Palindromes".

Exercise 9. (5 pts.) Read in Wikipedia what 'Referential transparency' means. Does Python have that feature? Can you spot a (counter)example in the course slides?

Exercise 10. (15 pts. ★) Solve the following points without using string methods.

a) Write a function `correct_spaces(s)` which, given a string s, returns a copy of the string where there are no two consecutive whitespaces. Example: `'welcome to  inner space '` gets corrected to `'welcome to inner space '`.

b) Write a variant of the function in the last point that *also* removes leading and trailing spaces. Example: `' space oddity  '` gets corrected to `'space oddity'`. You should not use the `strip` function.

Exercise 11. (25 pts. ★) Monte Carlo approximation of π.[1] In this exercise we will write a program to approximate the value of the mathematical constant π. First we should understand the method: Consider a circle of radius 1 which is embedded in a square of 2 by 2. I remind you

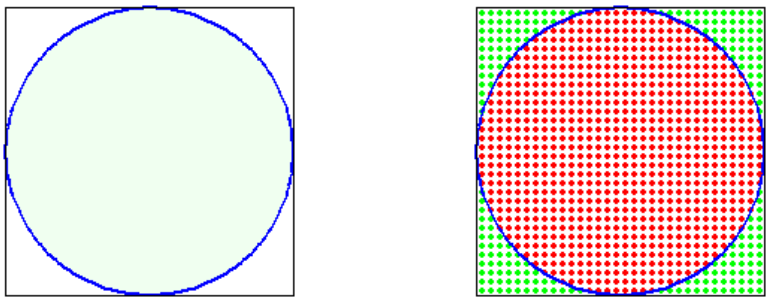

Figure 1: Square of 2×2, circle of radius 1 and grid.

that the formula to calculate the area of a circle is area $= \pi \times r^2$, therefore, in this case the area of the circle is π and the area of the square is 4. A simple way to approximate the area of the circle is to draw a grid of dots in the square and count how many of them are in and how many are out. In this example 812 points are inside the circle and 212 stand outside. Therefore,

$$\frac{812}{812+212} = 0.79296875$$

is an approximation of the area of the square covered by the circle. As the area of the square is 4, then the area of the circle is around $4 \times 0.79296875 = 3.171875$. That's an approximation of π!

We will do something similar, but using less points. We will randomly 'throw' points into the square (but with an uniform distribution, though). Some of them will be inside the circle and others outside. By an easy calculation we can show that the probability of a point falling inside the circle is $\frac{\pi}{4}$. Therefore, suppose we throw n points into the square and we get i inside. We can approximate the probability by this statistical simulation by computing i/n and thus getting $i/n \approx \pi/4$.

a) Write a function `dist(x,y)` that given two Floats representing a point in the plane, calculates the Euclidean distance between $(0,0)$ and (x,y). The `math` module may be useful.

b) Write a function `is_inside(x,y)` that given two Floats representing a point in the plane, returns `True` if the point is inside the circle of radius 1 centered in $(0,0)$ and `False` otherwise.

c) Write a function `napprox(n)` which throws n random points into the 2×2 square centered in $(0,0)$ and approximates π with that information. The `random` module may be useful.

d) Write a program, using the functions before, that asks the user for a number n and shows the n-approximation of π.

[1] Drawings were taken from `http://math.fullerton.edu/mathews/n2003/montecarlopimod.html`. Check the site for a deeper explanation of the Monte Carlo method for approximating π.

e) The precision of the Float data type is not that great. Modify your program to use the Decimal type provided by Python's `decimal` module. Do you think that there is a lot to change? Think twice.

Exercise 12. (15 pts.) The factorial function $! : \mathbb{N} \to \mathbb{N}$ is inductively defined as follows:

$$\begin{aligned} 0! &= 1 \\ n! &= (n-1)! \times n \end{aligned}$$

a) Write a function `rfact(n)` which calculates $n!$ using a recursive algorithm.

b) Write a function `ifact(n)` which calculates $n!$ using an iterative algorithm.

c) Which one is better? Hint: try them with $n > 900$. Explain what's happening.

d) Do a performance test using the `time` module. The user should be asked for a number of repetitions r. Then you should run each version r times and show how long it took for each algorithm. Feel free to check the Python documentation and Google if you don't know how to use the module to calculate time differences.

Exercise 13. (10 pts.) Write a *program* that, given the name of an existing plain text file as an argument, counts the number of words in it.[2]

Exercise 14. (20 pts. ★) Write a program that takes the name of a plain text file and a phrase as arguments. The program should open the file and look for the phrase. For every occurrence it should print the line number and the column number where it was found. If it is never found it should print "Sorry, phrase not found!".

[2]Plain should be understood as saved with notepad, gedit, etc. *Not* with Word, OpenOffice, etc.