Homework Set #1
(Warm-up)

Imperative Progamming with Python

January 2015

The homework should be uploaded using the BlackBoard system, it should *not* be printed. Read the `guidelines.pdf` (on the website) for the submission guidelines. **Due date:** 12/01/2015.

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

Exercise 2 (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 3 (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 4 (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 5 (10 pts.)**.** Do exercise 3 from Chapter 5 in the book: "Fermat's last theorem".

Exercise 6 (10 pts.)**.** Write a function `reverse(s)` which, given a string s, returns the reverse of s. Example: `'123456'` gets reversed to `'654321'`.

Exercise 7 (30 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 that the formula to calculate the area of a circle is area $= \pi \times r^2$,

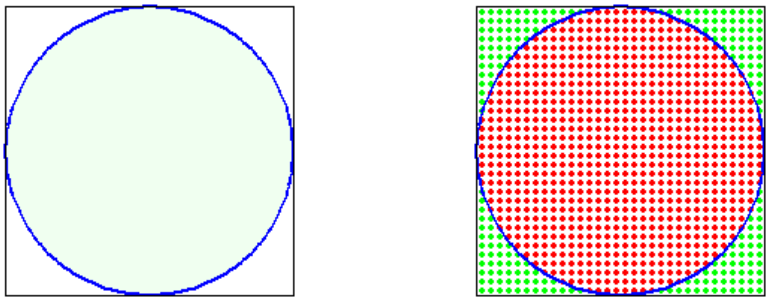

Figure 1: Square of 2×2, circle of radius 1 and grid.

therefore, in this case the area of the circle is π and the area of the square is 4. A simple way to

[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.

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 returns an approximation of π using that information. Check the `random` module.

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

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

Exercise 8 (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 9 (15 pts.)**.** *Cryptography: Caesar cipher.*[2]
In this exercise we will implement a method to encode text in such a way that only people knowing the key can decode it. This particular method is called 'Caesar cipher' and was developed by Julius Caesar who used it in his private correspondence. The method consist in taking the original text (called *plaintext*) and performing a 3-rotation on the English alphabet, obtaining the *ciphertext*. For simplicity, in this exercise we will only rotate the characters in the range 'a-z'. For example,

```
Plain:    ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-!#%1234567890
Cipher:   ABCDEFGHIJKLMNOPQRSTUVWXYZxyzabcdefghijklmnopqrstuvw-!#%1234567890
```

Observe that the characters which are not lowercase letters are not rotated. To decode, the rotation is simply done to the opposite direction. For example,

```
Ciphertext: qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald
Plaintext:  the quick brown fox jumps over the lazy dog
```

[2]More information in `http://en.wikipedia.org/wiki/Caesar_cipher`.

In this case the 'protection' of the method is that you need to know that you are performing a rotation, and that the offset of the rotation is the number 3.

Write a function `rotate(s, n)` which takes a string and an integer (that is, positive or negative number) and returns the n-rotation on the string. Remember to ignore the characters which are not lowercase letters. Example:

```
>>> rotate('the quick brown fox IS SUPER COOL!', -3)
'qeb nrfzh yoltk clu IS SUPER COOL!'
>>> rotate('qeb nrfzh yoltk clu IS SUPER COOL!', 3)
'the quick brown fox IS SUPER COOL!'
```

Your function should work properly for all integers n. The following functions may be useful:

- You can use the `in` predicate to test if a character (or string) occurs in another string.

  ```
  >>> 'd' in 'abcde'
  True
  >>> 'Q' in 'abcde'
  False
  ```

- You can use the `index` method to get the index (starting from 0) of a character in a string.

  ```
  >>> m = 'xzwabcde'
  >>> s = 'plaintext'
  >>> s[2]
  'a'
  >>> m.index(s[2])
  3
  ```