Lab 1 and homework 1:
- grades and comments posted on MyCourses

Biggest issues from the first two weeks:
- what is pseudo code?
- what is testing?

program-like description of the code (in English, structured as a program)

test all cases, especially special and typical cases
We will introduce:
- Fruitful functions (that return values)
- More advanced recursion and leap-of-faith approach
- Testing and debugging
- Pre- and post-conditions
- Basic loop (how to repeat a set of steps several times)
Fruitful functions

- Functions that return values, e.g. the area of a circle

\[ \text{Area of a circle of radius } r : \quad \pi r^2 \]

```python
def circleArea(r):
    return r*r*\pi

print(
circleArea(1))
x = circleArea(2)
```

test: with different values of r: e.g., 0, 1, 100, 250, possibly negative
Fruitful functions

- Another example: computing the distance between two points

\[ \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} \]

- Introduction to incremental development and debugging:

  see the book

[Sections 6.1 and 6.2]
Fruitful functions and recursion

- Example 1: Factorial

\[ n! = n \cdot (n-1) \cdot (n-2) \cdots 1 \]

\[
\begin{cases}
  n! = n \cdot (n-1)! & \text{if } n > 1 \\
  n! = 1 & \text{if } n = 1
\end{cases}
\]

- Stack-trace diagram:

```python
def fact(n):
    if n == 1:
        return 1
    if n > 1:
        tmp = n * fact(n-1)
        return tmp
```

[Section 6.5]
Fruitful functions and recursion

- Example 1: Factorial

```python
def fact(n):
    if n == 1:
        return 1
    if n > 1:
        return n * fact(n-1)
```

- “Leap of faith” approach to understanding recursive codes

```python
fact(n=5):
    n > 1 therefore we are going to return 5 * fact(4)
    we believe that this is computed as 24
    then we want to check that factorial(5) is computed correctly → OK, 120
```
Fruitful functions and recursion

- Example 2: Fibonacci

\[ a_n = a_{n-1} + a_{n-2} \quad \text{if} \quad n \geq 2 \]
\[ a_0 = 0 \]
\[ a_1 = 1 \]

def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    if n > 1:
        return fib(n-1) + fib(n-2)

- Stack-trace diagram:

DO: draw a complete stack-trace diagram for n=4
Fruitful functions and recursion

- Example 2: Fibonacci

```python
def fib0pt(n, last1, last2):
    if n == 0:
        return last1
    if n > 0:
        return fib0pt(n-1, last2, last1+last2)

call fib0pt(n, 0, 1)
```

- Leap of faith approach:

```
fib0pt(6, 0, 1)
```

Harder to apply here b/c we need to be able to say what should the func return given any three parameters.

However, we can do leap of faith on the original (inefficient) implementation.

- Question: is our pseudo code for Fibonacci efficient?

The original code NO, but the above is.

[Section 6.7]
The Tree Problem

Give a pseudo code that draws a tree as follows (for N=1 it is just a line of length S, for N=2 it is a small tree as shown with two additional branches of length S/2, etc.):

Idea: first we draw the line of length S then draw a smaller tree on the left (with N=1, S/2) and a smaller tree on the right.
The Tree Problem

Pseudo code, attempt 1:

```
function tree(N, S):
  if N > 1:
    draw a line of length S
    turn left by 45 deg.
    call tree(N-1, S/2)
    turn right by 90 deg.
    call tree(N-1, S/2)
    turn left 45 deg.
    draw a line of length S
    (i.e. go distance S backwards)
```

Post-condition:

- Finish at the same position
- and in the same direction as at the beginning

Problems:
- not in correct position to draw the right tree
- forget N=1 case

Trace of the pseudocode:
- `tree(N=2, S=100)`
  - `tree(N=1, S=50)`
    - `tree(N=1, S=25)`
      - `tree(N=1, S=12.5)`
        - `draw a line 50`
The Tree Problem

Testing of the pseudo code
The Koch Snowflake Problem

Give a pseudo code that draw a snowflake as follows: for N=1 it is a triangle with side-length S, for N=2 every straight line gets a "triangular bump" as shown, etc.

read the notes before lab
Pseudo code for the “sub-flake”:

```plaintext
function sub-flake (N, S):
    if N == 1:
        draw a line of length S
    if N > 1:
        draw a sub-flake (N-1, S)
        turn left by 60 deg.
        draw a sub-flake (N-1, S)
        turn right by 120 deg.
        draw a sub-flake (N-1, S)
        turn left by 60 deg.
        draw a sub-flake (N-1, S)
```

[Exercise 5.4]
Pseudo code for the entire snowflake:

```python
function snowflake(N, S):
    repeat three times:
        draw a subflake (N, S)
        turn right by 120 deg.
```

Testing:

test \( N = 0, 1, 2, 3 \), possibly 4

a range of \( S \)'s: e.g. 10, 20, 30 , also possibly 0 or negative values