# Solution to the exam: Introduction to programming (LT2111)

**Date: October 28, 2013, 9.00 – 12.00**

## Question 1 of 6: Summing the even numbers (3 points)

This can be done compactly with a list comprehension, or by using functions such as `filter`.

```
def sum_even(lst):
    return sum(n for n in lst if is_even(n))
```

Here is a slightly more verbose version:

```
def sum_even(lst):
    s = 0
    for n in lst:
        if is_even(n):
            s += n
    return s
```

## Question 2 of 6: Finding the longest word in a text file (4 points)

Again, this can be done with a one-liner or in a more step-by-step fashion. This task is easiest if you remember that if you want to read a file line by line, then it can be used as a sequence (like a list), so you can use `max` or `for`.

```
def find_longest_word(fn):
    with open(fn) as f:
        return max(f, key=len)[:-1]
```

or:

```
def find_longest_word(fn):
    longest = ""
    with open(fn) as f:
        for line in f:
            if len(line) > len(longest):
                longest = line
    return longest[:-1]
```

Here I removed the trailing end-of-line character. You won't lose points if you forgot that.

## Question 3 of 6: Lexical variety (4 points)

A short solution:

```
def ttr(words):
    return float(len(set(words))) / len(words)
```

. . . and another solution that is also OK:

```
def ttr(words):
    seen_words = {}
    N_t = 0
    for w in words:
        if w not in seen_words:
            seen_words[w] = True
            N_t += 1
    N_w = len(words)
    return float(N_t)/N_w
```

You will get a frown if you use a list to keep track of the words you have seen, but you won't lose any points.
Since the question does not say how you should handle the case where the input is empty, you are free to ignore that.
You **will** lose points if you don't handle the division properly, i.e. if your function returns zero when it shouldn't.

### Question 4 of 6: Readability (5 points)

If you consider it a sport to write compact programs, then you might have something like this:

```
def lix_score(text):
    nw = sum(len(sen) for sen in text)
    nlw = sum(len([w for w in sen if len(w) > 6]) for sen in text)
    return float(nw)/len(text) + 100*float(nlw)/nw
```

. . . but this is probably more readable:

```
def lix_score(text):
    ns = 0
    nw = 0
    nlw = 0
    for sen in text:
        ns += 1
        for w in sen:
            nw += 1
            if len(w) > 6:
                nlw += 1
    return float(nw)/ns + 100*float(nlw)/nw
```

### Question 5 of 6: Spam filter evaluation (6 points)

Here, you should assume that the class `SpamFilter` already exists, so you should just add one method to it. Here is a solution:

```
class SpamFilter(object):
    def __init__(self):
        ... do something ...
    def guess(self, text):
        ... do something ...

    def evaluate(self, ts):
        n = len(ts)
        ntt = 0
```

```
        nff = 0
        nft = 0
        for human_cat, text in ts:
            guess_cat = self.guess(text)
            if human_cat:
                if guess_cat:
                    ntt += 1
            else:
                if guess_cat:
                    nft += 1
                else:
                    nff += 1
        return float(ntt+nff)/n, float(nft)/(nft+nff)
```

The first five lines are here just to show you the context, and they are not mandatory in the solution.
If you implement `evaluate` as a separate function and not as a method, you will lose a few points. Also, you will be penalized if you forget the `self` input to the method, or if you call `guess` incorrectly.

## Question 6 of 6: Document similarity (8 points)

### (a, 4p)

You will first need to define a function that converts a document to a frequency table, and then `dot` is easy to implement. My solution looks like this:

```
from math import sqrt

def freqs(doc):
    d = {}
    for w in doc:
        if w not in d:
            d[w] = 1
        else:
            d[w] += 1
    return d

def dot(fs1, fs2):
    return sum(f1*fs2.get(w, 0.0) for w, f1 in fs1.iteritems())

def cos_sim(doc1, doc2):
    fs1 = freqs(doc1)
    fs2 = freqs(doc2)
    return dot(fs1, fs2)/sqrt(dot(fs1, fs1)*dot(fs2, fs2))
```

Note that I called `dot` with frequency tables as inputs. It's of course also OK if you call `freqs` from inside `dot`.

Here is a more elaborate implementation of `dot` that is also fine:

```
def dot(fs1, fs2):
    d = 0.0
    for w in fs1:
        if w in fs2:
            d += fs1[w]*fs2[w]
    return d
```

**(b, 4p)**

To get the `n` most similar documents, we need to sort the documents by similarity to `d`, and then take the `n` first of them. Note that `cos_sim` gives a value of 1 if two documents are identical and 0 if they are completely different, so we need to sort in decreasing order (`reverse=True`). Here, the slightly tricky issue is how to carry out the sorting. I think the easiest solution is what Python programmers like to call "decorate-sort-undecorate":

```
def most_similar(d, n, docs):
    result = sorted(((cos_sim(d, d2), d2) for d2 in docs), reverse=True)[:n]
    return [d2 for _, d2 in result]
```

Here is a step-by-step solution:

```
def most_similar(d, n, docs):
    scored_docs = [ (cos_sim(d, d2), d2) for d2 in docs]
    sorted_docs = sorted(scored_docs, reverse=True)
    toplist = sorted_docs[:n]
    return [d2 for (sim, d2) in toplist]
```

You won't lose points if you forget the "undecoration" step, i.e. if you return a scored list of documents rather than just the documents.