Exam: Introduction to programming (LT2111)

Date: December 16, 2013, 9.00 – 12.00

Course responsible Richard Johansson, Språkbanken, Department of Swedish

Exam accessories None

Grade limits Pass with distinction: 24p, Pass: 15p, Max: 30p

Please note:

- If there is something you don't understand about a question, please ask the course responsible when he comes to the exam room.
- Write legibly (unreadable = wrong)!
- Make sure that your code has a clear indentation (if it is unclear, then the least favorable interpretation will be chosen).
- Number the pages, and start every question on a new page.
- Points will be removed for unnecessarily complicated or unstructured solutions.
- You may use built-in functions and methods, unless otherwise stated, but make sure that you know what they do. If you are unsure, define your own functions.
- If your solution to a question is only partial, please turn it in anyway! Every point counts.

Question 1 of 6: Summing numbers (4 points)

Define a function `sum_less_than_ten(lst)` that computes the sum of those numbers in a list `lst` that are less than 10. Here are a few examples of the output of `sum_less_than_ten`.

```
>>> sum_less_than_ten([7, 8, 13, 27, -6, 1])
10
>>> sum_less_than_ten(range(20))
45
>>> sum_less_than_ten([])
0
```

Question 2 of 6: Fixing errors (5 points)

The functions listed in (a) and (b) contain one or more errors and will print exceptions when run. Correct all errors in the functions so that they run as intended.

(a) A function to sum some numbers that we have read from a file:

```
def sum_the_numbers(numbers):
    output = 0
    for n in numbers:
        output += n
    return output
```

Here is how the function should be called:

```
print sum_the_numbers(["1", "77", "3", "8"])
```

Here is the result:

```
TypeError: unsupported operand type(s) for +=: 'int' and 'str'
```

(b) A function to build a frequency table for the words in a corpus:

```
def make_frequency_table(wordlist):
    table = {}
    for word in words:
        table[word] += 1
    return table
```

Here is how the function should be called:

```
print make_frequency_table(["this", "is", "a", "corpus", "in", "a", "list"])
```

Result:

```
NameError: global name 'words' is not defined
```

Question 3 of 6: Lexicon coverage (5 points)

Define a function `lexicon_coverage(lexicon, filename)` that determines the proportion of the words in a file that are listed in a lexicon. The lexicon is represented as a Python list. The file consists of one word per line. For instance, if we process the text file

```
this
is
a
short
file
with
some
words
in
it
```

and the lexicon is `['a', 'an', 'are', 'be', 'in', 'is', 'it', 'on', 'the']`, 4 out of the 10 words in the file are found in the lexicon, so the return value should be the proportion $4/10 = 0.4$.

Question 4 of 6: The most frequent word (5 points)

Define a function `most_frequent_word(words)` that returns the most frequent word in a corpus. If the highest frequency is F and there are several words whose frequency is F, then return any of those words. The corpus is represented as a list of strings. For example, when running this code

```
print most_frequent_word(["this", "is", "a", "corpus", "in", "a", "list"])
```

the program should print the word `a`.

Question 5 of 6: Bank accounts (5 points)

Implement two classes `Bank` and `BankAccount`. The class `Bank` should have two methods:

- `open_new_account`, which creates a new account for a customer, with a given initial amount of money;
- `get_account`, which returns the account (a `BankAccount` object) for a customer.

The class `BankAccount` should have three methods:

- `get_balance`, which returns the amount of money currently stored in the account;
- `deposit`, which adds an amount of money to the account;
- `withdraw`, which takes an amount of money from the account. This method should protest if you want to withdraw more money than the account contains, for instance by raising an exception: `raise ValueError(error_message)`.

Here is an example program using these classes:

```
the_bank = Bank()
my_name = "Alice Bob"
the_bank.open_new_account(my_name, 1000)
my_account = the_bank.get_account(my_name)
my_account.withdraw(600)
my_account.deposit(100)
print my_account.get_balance()
my_account.withdraw(600)
```

The result of running this program should be something like

```
500

... ValueError: you don't have that much money!
```

Question 6 of 6: Part-of-speech tagging (6 points)

Implement a program consisting of two functions: first a function `train_tagger` that trains a part-of-speech tagger on a training corpus, and then a function `run_tagger` that uses it to assign part-of-speech tags to the words in an untagged corpus.
Here is an example of how the functions can be used:

```
tagger = train_tagger("training_corpus_file")
run_tagger(tagger, "untagged_corpus_file")
```

(a) In the function `train_tagger`, your program goes through a training corpus. This is a text file where each line is a word/tag pair. The word and the tag are separated by a space. Here is an example:

```
This      PRONOUN
is        VERB
a         DETERMINER
file      NOUN
and       CONJUNCTION
the       PUNCTUATION
file      NOUN
contains  VERB
some      DETERMINER
text      NOUN
.         PUNCTUATION
Please    INTERJECTION
file      VERB
it        PRONOUN
in        PREPOSITION
the       DETERMINER
drawer    NOUN
.         PUNCTUATION
```

Your function `train_tagger` should determine for each word what is the most frequent part-of-speech tag for that word. In the example file, the word *file* occurs twice as a `NOUN` and once as a `VERB`, so your program determines that the most frequent tag for this word is `NOUN`.

Furthermore, your function should find which is the most common tag in general in the training corpus. In the example, this is `NOUN`, which occurs four times.

The function should return 1) a dictionary containing the most frequent tag for each observed word, and 2) the most frequently observed tag in general.

(b) After we have called `train_tagger`, we can use its result to assign tags to words in texts that we haven't seen before. Write a function `run_tagger` that goes through a text file with one word per line, and assigns a tag for each word. Here is an example of such a file:

```
This
is
a
new
file
.
```

For each word occurring in the text file, your program should print the most frequent tag in the training file for that word. If a word occurs that you haven't observed in the training file, then print the most frequent tag in general. For the example, this means that your program will print

```
PRONOUN
VERB
DETERMINER
NOUN
NOUN
PUNCTUATION
```