Jonas van den Brink, jvbrink

j.v.d.brink@fys.uio.no

4.1

```
#!/usr/bin/env python
import sys, re

usage = 'Usage: %s infile word' % sys.argv[0]

# Try reading cmd-line arguments
try:
  infile = sys.argv[1]
  word = sys.argv[2]
except:
  print usage; sys.exit(1)

# Read in text as a single string
infile = open(infile, 'r')
text = "".join(infile.readlines())
infile.close()

# Read flags
b = '-b' in sys.argv # Respect word boundaries
i = '-i' in sys.argv # Ignore letter case

# Create expression and compile pattern
expr = r'\b'+word+r'\b' if b else word
pattern = re.compile(expr, re.I) if i else re.compile(expr)

# Find number of occurances
count = len(re.findall(pattern, text))

# Print results
b = 'word' if b else 'string'
i = ' (case insensitive)' if i else ''
print "Number of occurances of %s '%s'%s: %i" % (b, word, i, count)

'''
user$ python count_words.py football.txt ball
Number of occurances of string 'ball': 3
user$ python count_words.py football.txt goal -i
Number of occurances of string 'goal' (case insensitive): 3
user$ python count_words.py football.txt goal -i -b
Number of occurances of word 'goal' (case insensitive): 2
user$ python count_words.py football.txt goal -b
Number of occurances of word 'goal': 1
'''
```

4.2

The regex pattern used is as follows

```
([+\-]?\d+\.?\d*|[+\-]?\.\d+|[+\-]?\d\.\d+[Ee][+\-]\d\d?)
```

This expressions matches to three different structures of numbers. First there is

```
[+\-]?\d+\.?\d*
```

Here, `[+\-]?` matches either a '+' or a '-', or nothing, `\d+` matches any decimal digit one or more times, `\.?` is an optional match to '.' and `\d*` matches any decimal digit zero or more times. This expression is fine. Next we have:

```
[+\-]?\.\d+
```

Again `[+\-]?` matches either a '+' or a '-', or nothing, while `\.` matches to a '.' and `\d+` matches any decimal digit one or more times. This part is also fine. Finally we have

```
[+\-]?\d\.\d+[Ee][+\-]\d\d?
```

This structure is supposed to match any number on scientific form. Again we start of with the optional sign: `[+/-]?`, followed by a single decimal digit and a '.': `\d\.`, note that in scientific form all numbers are written on this form, so everything is fine so far. Next we match one or more decimal digits and either an 'E' or 'e': `\d+[Ee]'`, we then include the sign of the expoential`[+\-]` followed by exactly two decimal digits: `\d\d`. There are three problems with this expression, first of, there is no requirement of having any digits between the '.' and the exponential in a scientific form, so we replace `\d+` with `\d*`. Second, it is perfectly okey to write a scientific form without the positive sign in the expoential, so we put in an optional character: `?`. Lastely, there is no need to have exactly two decimal digits in the exponential, so we change to match one or more digits: `\d+`.

The final regex expression we use is then

```
([+\-]?\d+\.?\d*|[+\-]?\.\d+|[+\-]?\d\.\d*[Ee][+\-]?\d+)
```

And the full program is as follows

```
#!/usr/bin/env python
"""Find all numbers in a string."""
import re
r = r"([+\-]?\d\.\d*[Ee][+\-]?\d+|[+\-]?\d+\.?\d*|[+\-]?\.\d+)"
c = re.compile(r)
s = "an array: (1)=3.9836, (2)=4.3E-09, (3)=8766, (4)=.549"
numbers = c.findall(s)
# make dictionary a, where a[1]=3.9836 and so on:
a = {}
for i in range(0,len(numbers)-1,2):
    a[int(numbers[i])] = float(numbers[i+1])
sorted_keys = a.keys(); sorted_keys.sort()
for index in sorted_keys:
    print "[%d]=%g" % (index,a[index])

'''
user$ python regexerror.py
[1]=3.9836
[2]=4.3e-09
[3]=8766
[4]=0.549
'''
```

4.3

There are several reasons why these expressions fail. First of, they match to any character in the groups by using `(.+)`, but we would like to only get the numbers, so we should instead match only decimal digits inside the groups, so we instead match our group as `(\d+)` and put an optional number of whitespace characters on each side of the groups. The first expression also has an optional extra character where it makes no sense to use one, so we remove this.

The final regex expression we use is then

```
\[\s*(\d+)\s*:\s*(\d+)\s*,?\s*(\d*)\s*\]
```

And the full program is as follows

```
loop1 = '[0:12]'      # 0,1,2,3,4,5,6,7,8,9,10,11,12
loop2 = '[0:12, 4]'   # 0,4,8,12
r = r'\[\s*(\d+)\s*:\s*(\d+)\s*,?\s*(\d*)\]'
r = r'\[\s*(\d+)\s*:\s*(\d+)\s*,?\s*(\d*)\s*\]'
import re
print re.search(r, loop1).groups()
print re.search(r, loop2).groups()

'''
user$ python loop_regex.py
('0', '12', '')
('0', '12', '4')
'''
```

4.4

The regex expression used here does what is wanted, it is more the use of the `re.findall`-function that is a bit strange. We can easily extract the lower and upper bounds from the string by using the `re.finditer`-function, which then returns an iterator of `MatchObject`-instances. We can then loop over the iterator and extract the `number`-group for each match.

```
import re

real = r"\s*(?P<number>-?(\d+(\.\d*)?|\d*\.\d+)([eE][+\-]?\d+)?)\s*"
pattern = re.compile(real)

some_interval = "[3.58e+05132 , 6E+09]"

# Create an iterator over MatchObject instances
matches = pattern.finditer(some_interval)

# Loop over iterator and extract 'number'-groups
boundaries = [m.group('number') for m in matches]

# Print out resulting boundaries
print '\n'.join(boundaries)

'''
user$ python findallerror.py
3.58e+05132
6E+09
'''
```