1)
```
hand = open('mbox-short.txt')
for line in hand:
        line = line.rstrip()
        if re.search('From:',line):
          print(line)
```

2)
```
import re
hand = open('mbox-short.txt')
for line in hand:
        line = line.rstrip()
        if re.search('^From:', line):
          print(line)
```

3)
```
import re
hand = open('mbox-short.txt')
for line in hand:
      line = line.rstrip()
      if re.search('^F..m:', line):
        print(line)
```

4)
```
import re
hand = open('mbox-short.txt')
for line in hand:
        line = line.rstrip()
        if re.search('^From:.+@', line):
          print(line)
```

5)
```
import re
hand = open('mbox-short.txt')
for line in hand:
      line = line.rstrip()
        x = re.findall('\S+@\S+', line)
        if len(x)>0:
          print(x)
```

Problem Solving Using Python and R Lab

Lab8. Python Regular Expressions

Question1. Using Email Collections file, mbox-short.txt, write a python program for the following queries.

1. Search for lines that contain 'From' and print them
2. Search for lines that start with 'From' and print them
3. Search for lines that start with 'F', followed by 2 characters, followed by 'm:'
4. Search for lines that start with From and have an at sign and print them
5. Search for lines that have an at sign between characters and print them (Use findall())
6. Search for lines that have an at sign between characters. The characters must be a letter or number and print them
7. Search for lines that start with 'X' followed by any non white space characters and ':', followed by a space and any number. The number can include a decimal.
8. Search for lines that start with 'X' followed by any non whitespace characters and ':' followed by a space and any number. The number can include a decimal. Then print the number if it is greater than zero.
9. Search for lines that start with 'Details: rev=', followed by numbers and '.'. Then print the number if it is greater than zero
10. Search for lines that start with From and a character followed by a two digit number between 00 and 99 followed by ':'. Then print the number if it is greater than zero

6)
```
import re
hand = open('mbox-short.txt')
for line in hand:
        line = line.rstrip()
         x = re.findall('[a-zA-Z0-9]\S+@\S+[a-zA-Z]',
                   line)
         if len(x) >0:
           print(x)
```

7)
```
import re
hand = open('mbox-short.txt')
for line in hand:
          line = line.rstrip()
          if re.search('^X\S*: [0-9.]+', line):
```

8)
```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X\s*: ([0-9.]+)', line)
    if len(x) >0:
     print(x)
```

9)
```
import re
hand = open('mbox-short.txt')
for line in hand:
    Line = line.rstrip()
     x = re.findall('^Details:.*rev=([0-9.]+)', line)
     if len(x) >0:
         print(x)
```

10)
```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
     x = re.findall('^From.*([0-9][0-9]):', line)
     if len(x) >0:
       print(x)
```

Question2. Baby Names Popularity Analysis

Reference: https://developers.google.com/edu/python/exercises/baby-names

The Social Security administration has this neat data by year of what names are most popular for babies born that year in the USA. The files baby1990.html baby1992.html ... contain raw html pages. Take a look at the html and think about how you might scrape the data out of it.

In the babynames.py file, implement the extract_names(filename) function which takes the filename of a baby1990.html file and returns the data from the file as a single list -- the year string at the start of the list followed by the name-rank strings in alphabetical order. ['2006', 'Aaliyah 91', 'Abagail 895', 'Aaron 57', ...].

Modify main() so it calls your extract_names() function and prints what it returns (main already has the code for the command line argument parsing). If you get stuck working out the regular expressions for the year and each name, solution regular expression patterns are shown at the end of this document. Note that for parsing webpages in general, regular expressions don't do a good job, but these webpages have a simple and consistent format.

Rather than treat the boy and girl names separately, we'll just lump them all together. In some years, a name appears more than once in the html, but we'll just use one number per name. Optional: make the algorithm smart about this case and choose whichever number is smaller.

Build the program as a series of small milestones, getting each step to run/print something before trying the next step. This is the pattern used by experienced programmers -- build a series of incremental milestones, each with some output to check, rather than building the whole program in one huge step.

Printing the data you have at the end of one milestone helps you think about how to re-structure that data for the next milestone. Python is well suited to this style of incremental development. For example, first get it to the point where it extracts and prints the year and calls sys.exit(0). Here are some suggested milestones:

- Extract all the text from the file and print it
- Find and extract the year and print it
- Extract the names and rank numbers and print them
- Get the names data into a dict and print it
- Build the [year, 'name rank', ...] list and print it
- Fix main() to use the ExtractNames list

Earlier we have had functions just print to standard out. It's more re-usable to have the function *return* the extracted data, so then the caller has the choice to print it or do something else with it. (You can still print directly from inside your functions for your little experiments during development.)

Have main() call extract_names() for each command line arg and print a text summary. To make the list into a reasonable looking summary text, here's a clever use of join: text = '\n'.join(mylist) + '\n'

Question 2:

```
import re
import sys

names = [ ]
f = open(filename, 'r')
text = f.read()
year_match = re.search(r'popularity\sin\s(\d\d\d\d)',
                        text)
if not year_match:
    sys.stderr.write('Couldn\'t find the year!\n')
    sys.exit(1)
  year = year_match.group(1)
  names.append(year)
tuples = re.findall(r'<td>(\d+)</td></td>
              (\w+)</td>\<td>(\w+)</td>',
              text)
names_to_rank = {}
for rank_tuple in tuples:
  (rank, boyname, girlname) = rank_tuple
  if boyname not in names_to_rank:
    names_to_rank[boyname] = rank:
  if girlname not in names_to_rank:
    names_to_rank[girlname] = rank
sorted_names = sorted(names_to_rank.keys())
```

The summary text should look like this for each file:

```
2006
Aaliyah 91
Aaron 57
Abagail 895
Abbey 695
Abbie 650
...
```

```
for name in sorted_names:
    names.append(name + " " + names_to_rank[name])
  return names

  extract_names('baby2006.html')
```