You have already learned about the following concepts:

- `for` loops over strings and sequences of numbers
- the accumulator pattern
- comparison operators (greater than, less than, etc.)
- the `bool` data type (values: `True`, `False`; operators: `and`, `or`, `not`)
- `if` statements

This handout presents several problems where solving them requires synthesizing several of these concepts into a single program. The paper version of the handout states only the problems; *another version of the handout will be posted on moodle with solutions*.

1 Accumulator pattern (review)

The **accumulator pattern** is a general strategy for completing a task using a for loop. We can use it to complete tasks involving strings where we accumulate a result by processing each character of the string one at a time. Here are some key steps in applying the accumulator pattern:

1. Answer these key questions:
   (a) What do you want to compute? What is the final result?
   (b) Assuming the final result is a data value. What is its type?
   (c) How can we build up the final result by processing one letter at a time?
2. Initialize an accumulator variable: make sure it has the right type!
3. Each time through the loop, update the accumulator variable.
4. When the loop finishes, the accumulator variable should contain the final result.

For example, this computes the length of a string by accumulating a count.

```
# goal: print reversed copy of user's name
name = raw_input("What is your name? ")
rev_name = ''     # accumulator variable will hold reversed name
for letter in name:
    rev_name = letter + rev_name # update accumulator
# when loop is over, rev_name has reversed name!
print "Your name in reverse:", rev_name
```

2 Conditional accumulators

A **conditional accumulator** uses the accumulator pattern but does not necessarily treat each item in the sequence (i.e., each character in the string) the same. What it does is *conditional* on the value of the item (i.e., character). In other words, you need to have an `if` statement inside the `for` loop.

1. Write a program that asks the user to type a phrase and then reports the number of times the letter "e" appears in the phrase. As an extra challenge, make your program case-insensitive: the program reports the number of times the letter appeared, regardless of whether it appeared as "e" or "E."
2. Write a program that asks the user for a phrase and *disemvowels* it (i.e., prints the phrase with all the vowel letters removed). For example, if the user enters the phrase "The quick brown fox jumps over the lazy dog" the program would print "Th qck brwn fx jmps vr th lzy dg."

3 Search

There are many natural problems that can be characterized as **search** problems. Here, we look at search over sequences. The basic idea is that we are given a sequence (say, a string of characters typed by the user) and we want find out whether a particular pattern occurs in that sequence. This kind of search problem is solved by using a loop over the sequence with variables that keep track of whether or not you have found the item you are looking for.

3. Write a program that asks the user for two things: a phrase and a single letter. It then reports whether or not that letter occurs in the phrase. As an extra challenge, solve this problem without using a single `int` value – i.e., don't count the occurrences of the letter. Examples:

```
Enter a phrase: Hello, world!
Enter a single letter: w
The letter w occurs in the phrase: 'Hello, world!'

Enter a phrase: Hello, world!
Enter a single letter: z
The letter z does not occur in the phrase: 'Hello, world!'
```

4. Write a program that asks the user for a phrase and then finds the largest letter in it. Remember that "A" is the *smallest* letter, since capital letters are considered "less than" lowercase letters. You can assume that the phrase contains at least one letter and that letter will be at least as large as "A." Thus, "A" is a good initial guess for the largest letter.
5. Write a short program that asks the user for a phrase and then prints whether the string contains an "i" before an "e." The "e" does **not** need to immediately follow the "i." For example, both "goodgrief" and "listen" contain an "i" before an "e." (You can assume that the strings entered are in lowercase.)