Solve Problems A-C on paper. Solve Problems D-F in a single Python file `regexp.py`. Your code should include comments to explain any obscure or tricky bits. It should also include demonstration code. Hand in `regexp.py` electronically, by dropping the file in your hand-in folder on the COURSES file server.

A. Working over $\Sigma = \{a, b, c, d\}$, draw an NFA that recognizes the same language as the regular expression

$$(ad \cup b \cup c)^*(dda)^* \cup ac.$$

B. Let's extend our concept of regular expressions to include another classic regular operation: complementation. For any regular expression α, we define $\sim \alpha$ to be a regular expression describing the language $L(\sim \alpha) = \Sigma^* - L(\alpha)$. Working over the alphabet $\Sigma = \{a, b\}$, find a regular expression equivalent to the pattern $\sim (a^*)$.

C. The *Hamming distance* $H(w, x)$ between two bit strings w and x is defined as follows. If $|w| \neq |x|$, then $H(w, x) = \infty$. If $|w| = |x|$, then $H(w, x)$ is the number of bits in which w and x differ. For example, $H(00010, 10111) = 3$. For any set A of bit strings, define $N_2(A)$ to be the set of bit strings within Hamming distance 2 of A:

$$N_2(A) = \{w : \exists x \in A \text{ such that } H(w, x) \leq 2\}.$$

Prove that if $A \subseteq \{0, 1\}^*$ is regular, then so is $N_2(A)$. (Hint: If $A = L(M)$, where M has states Q, then construct an NFA with states $Q \times \{0, 1, 2\}$. Use the extra state information to track how many "errors" have occurred thus far.)

D. Come up with a Python regular expression that describes mis-capitalized words. For the sake of this problem, a word is said to be mis-capitalized if it consists of two or more letters and any letter after the first one is upper-case. You may assume that only alphabetical characters appear in words. For example, when I feed your regular expression to

```
re.findall(yourregexp, ’This is okay tHis IS nOT Okay.’)
```

I should get a result of `[’tHis’, ’IS’, ’nOT’]`.

E. In this problem we learn how to harvest e-mail addresses from texts such as web pages. You must promise never to use this power for evil.

An e-mail address such as `supersnake@carleton.edu` consists of a local part, `supersnake`, and a hostname, `carleton.edu`. The local part is a string made of one or more characters from this set: upper- and lower-case English letters, the digits `0` through `9`, the characters `!`, `#`, `$`, `%`, `&`, `’`, `*`, `+`, `-`, `/`, `=`, `?`, `^`, `_`, `‘`, `{`, `|`, `}`, `~`, and the period `.`. The period is allowed to be neither the first nor the last character in the local part. The hostname is a string made of one or more characters from this set: lower-case English letters, the digits `0` through `9`, the period `.`, and the hyphen `-`. The local part and the hostname are separated by a single `@`. (There are a few more rules to real e-mail addresses, but this is good enough for our purposes.)

Design a Python regular expression that matches e-mail addresses as just specified.

F. I like to write my dates in the format `yyyy/mm/dd`, but sometimes I accidentally write `mm/dd/yyyy` because that's how I was raised. Write a Python function `fixdates` that takes a string as input, uses a regular expression to fix all dates in the string to my liking, and outputs the fixed string. You may assume that all years are four-digit — I'm Y2K-compliant — but remember that months and days can be one- or two-digit. (Hint: You need to use substitutions and multiple groups.)