CS 97 Discussion 2 Practice Problems

Regular Expressions

For the following exercises, quickly verify your answers using `grep` as you go. In addition, assume the environment locale variable is set to `LC_ALL=C` (i.e. letters are ASCII a-zA-Z only).

1. Write a **basic** regular expression that causes `grep` to match lines ending in the characters “dog”.

   It should match:
   - “bulldog”
   - “smol dog”

   It should not match:
   - “doggo”
   - “big dogs”

2. Write a **basic** regular expression that causes `grep` to match lines that contain the subsequence “abc”. A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e. "bce" is a subsequence of "abcde" while "bec" is not).

   It should match:
   - “aQbQcQ”
   - “abacus”
   - “QQaaQbQQQcQQQ”
   - “abc”

   It should not match:
   - “bac”
   - “QQQacbQQQabbbb”
   - “blockade”

3. Write a **basic** regular expression that causes `grep` to match lines solely consisting of alphabetical characters (A-Za-z) and digit characters (0-9), where no alphabetical characters appear before digit characters.

   It should match:
   - “0123456789abc”
   - “42WallabyWay”
   - ""
   - “Meow”
   - "1234"

   It should not match:

- "abc123"
- "1$7 apples"
- "1a2b3c"

4. Write an **extended** regular expression that causes `grep -E` to match lines that either contain at least one digit or the string “flower”. It should be as short as possible. The optimal solution is 12 characters long. (If you like this sort of thing, check out code golfing!)

More Shell Scripting

1. (*based on a true story*) You’re a freelance mobile developer creating a stickers app for a client. You’re creating a screen that shows all of the stickers at once to the user; to do this you need to load each sticker image from disk. Stickers can have three different sizes (small, medium, large), and you *specifically* tell their graphic designer to send you image files using the naming convention:

   “`<size>-<sticker name>.<file extension>`”
   - “small-dog.png” and “medium-fish.jpeg” are examples of valid file names that will work with the code you’ve already written.

   However, you somehow still receive thousands of image files in the format:

   “`<sticker name>[0|1|2].<file extension>`”
   - `[0|1|2]` means the character “0” or “1” or “2”. The designer decided to make “0” represent “small”, “1” represent “medium”, and “2” represent “large”.
   - Instead of “small-dog.png”, you received “dog0.png”.
   - Instead of “medium-fish.jpeg”, you received “fish1.jpeg”.
   - `<sticker name>` and `<file extension>` never contain anything other than alphabetic characters.

   You *could* rewrite your app’s source code, but let’s assume that you’ve already set up an entire infrastructure that relies on the files being named in the format you wanted them to be. It’s obviously infeasible for you to rename thousands of files manually, but luckily, you took CS ~~35L~~97 and are a shell scripting pro. Fill in the blanks in the following shell script that you’ll run on a directory containing all the image files to make up for other people’s incompetence.

   In addition, for the line with `# !!!` after it, answer the following questions:
   a. Why do we need to use `echo $FILE` instead of `< $FILE`?
   b. What does the command `tr -cd '012'` do?

```
#!__________

DIRECTORY=$1
FILES=`ls $DIRECTORY`

for FILE in ________
do
  NUMBER_SIZE=`echo $FILE | tr -cd '012'` # !!!

  if [ $NUMBER_SIZE = 0 ]
  then
    SIZE=_____________
  elif __________________
  then
    SIZE=_____________
  else
    SIZE="large"
  fi

  NEW_FILENAME=`echo $FILE | sed ___________________________`
  # Rename the file (remember we’re one directory above…)
  _____________________________________________
done
```