Lab 10 - Python and Regular Expressions

The files for this final lab are as usual located in:

`/home/zastre/seng265/lab-10/`

Space does not permit a full treatment of the regular expression language and its implementation in Python. A friendly introduction to the language and its use can be found at the Python Regular Expression HOWTO document.

Exercise A

This is a code-reading exercise. We have an input file named `dpkg.log` that contains records of installed software, one software package per line (dpkg is a popular tool for managing installed software in Linux.)

A typical portion of this file (lines 14 - 22) is shown here:

```
2020-07-07 16:06:10 status installed linux-base:all 4.5ubuntu1.2
2020-07-07 16:06:10 trigproc man-db:amd64 2.8.3-2ubuntu0.1 <none>
2020-07-07 16:06:10 status half-configured man-db:amd64 2.8.3-2ubuntu0.1
2020-07-07 16:06:10 status installed man-db:amd64 2.8.3-2ubuntu0.1
2020-07-08 13:05:03 startup archives unpack
2020-07-08 13:05:03 upgrade libc-bin:amd64 2.27-3ubuntu1 2.27-3ubuntu1.2
2020-07-08 13:05:03 status half-configured libc-bin:amd64 2.27-3ubuntu1
2020-07-08 13:05:03 status unpacked libc-bin:amd64 2.27-3ubuntu1
2020-07-08 13:05:03 status half-installed libc-bin:amd64 2.27-3ubuntu1
```

We would like a list of software package names, but only if that software is fully 'installed', preceded by the line number on which that package appears. Our goal is therefore to parse this file, producing output that looks more like this (for the subset of input lines above):

```
14: linux-base
17: man-db
```

The four versions of the `installed` script show four increasingly effective attempts to do this, using regular expressions. Run each of the versions with `dpkg.log` provided to stdin to see the results.

- `installed.py` simply attempts to detect the word "installed", followed by a space, then some text, then a space and more text.

- The first version of `installed.py` unfortunately matches "half-installed" as well as "installed", which gives us too many matches. `installed2.py` attempts to address this problem, by insisting that a space must appear before the word.
- `installed3.py` makes a first attempt at stripping out the unnecessary information after the package name (the ":amd64" architecture text).

- `installed4.py` finishes the work by extracting the correct group from the match object generated by search. It also improves performance and program flow by pre-compiling the pattern.

Exercise B

`installed-ondate.py` continues on from Exercise A. Our goal this time is to be able to specify a date as a command line argument, and print only those packages that were 'installed' on that date. (We'll precede the package name with the date, rather than the line number from the log this time).

For example, if we run the program like this:

```
python3 installed-ondate.py 2020-07-22 < dpkg.log
```

our output should be identical to that provided in the file `ondate-2020-07-22.txt`:

```
2020-07-22: linux-image-4.15.0-109-generic
2020-07-22: linux-modules-4.15.0-109-generic
2020-07-22: libpython2.7-minimal
2020-07-22: libpython2.7-stdlib
2020-07-22: python2.7-minimal
2020-07-22: python2.7
2020-07-22: mime-support
2020-07-22: man-db
2020-07-22: libpython3.6-minimal
2020-07-22: libpython3.6-stdlib
2020-07-22: python3.6-minimal
2020-07-22: libpython3.6
2020-07-22: python3.6
2020-07-22: libc-bin
2020-07-22: man-db
2020-07-22: mime-support
```

You do NOT need to use the datetime module to solve this problem - assume that the date provided as an argument is always valid, and is formatted as YYYY-MM-DD (the same format as in the `dpkg.log` file). In other words, simple string equality will be fine.

Once you have completed this, try `installed-range.py`. This program should produce the same type of output as above, but in this case for all days within an inclusive range. For example,

```
python3 installed-range.py 2020-07-15 2020-07-16 < dpkg.log
```

should produce the output shown in `range-2020-07-15-to-2020-07-16.txt`. Again the datetime module is not needed, since dates expressed in YYYY-MM-DD order are naturally ordered lexicographically, and therefore can be distinguished by $<$ and $>$ operators.

Exercise C

This is a code-reading exercise. Sometimes when we find a match with our pattern, we wish to substitute new data in place of the matched text. The `subbing.py` program does so.

Exercise D

The program `gen-messages.py` is intended to create text files - letters to subscribers of “Unix Users Quarterly” telling them that their subscriptions have lapsed. The expected output is in `_messages`.

Each subscriber should get their own text file, all created in a directory called `messages`. Each subscriber also has a line in `data.csv` - a typical line looks like this:

```
Ashbourne,Lorraine,February 22 2020,fwiw1691
```

This line has comma separated fields indicating the subscriber’s last name, first name, expiry date, and required filename.

You should base your solution on the provided `gen-messages.py` file. You’ll need to modify the template string (which currently contains nonsense) - use one of the example output text files as a guide.

You’ll then need to modify the for-loop in main - split up the line, and substitute the strings from the `data.csv` line into the template text.

Finally, write the resulting string to the appropriate location, with the correct filename. Once you have done so, your `messages` and `_messages` folders should be identical.

Thank You

Make sure your TA records your participation in this lab. This is the final lab of the semester. The SENG265 teaching team thanks you for your hard work!