Assignment 2. Lisp and Python scripting

[97 home > assignments]

Do this assignment on the SEASnet GNU/Linux servers `lnxsrv06`, `lnxsrv07`, `lnxsrv09`, or `lnxsrv10`, with `/usr/local/cs/bin` prepended to your `PATH`.

If you need a hint, ask a TA or an LA. This assignment is not intended to be done without any hints.

Laboratory: Lisp scripting

- Robert J. Chassell, An Introduction to Programming in Emacs Lisp (2020)

Exercise 2.1: Navigating through Emacs source code

The basic idea here is to get a mental model of how Emacs works by looking at a bit of its keybindings and source code.

Start up a fresh Emacs with a `*scratch*` buffer.

To warm up, compute 2^{3^4} (i.e., 2**(3**4)) in the *scratch* buffer, by using the `expt` function. Use Emacs to determine whether this number fits into a 64-bit signed integer, by writing a Lisp expression that yields `t` if so and `nil` if not.

Type `M-:` and use it to compute 2^{3^4}.

Get a list of keybindings by typing `C-h b`.

Look for two keybindings: `C-h k` and `M-SPC`. `C-h k` stands for “Type Control-h, then ‘k’.” `M-SPC` is “Meta Space”; on good keyboards you can get this by holding down Alt while hitting the space bar, but you may need to type “Esc” and then follow by hiting the space bar. We will examine these two keybindings in more detail.

Type `C-h k C-h k` and describes what happens and why. (This should relate to the `C-h b` output mentioned previously.)

Type `C-h k M-SPC` and describes what happens and why. (This should also relate.)

Try out `M-SPC` on some sample text, to see how it works.

Visit the source code for the function that implements `M-SPC`, by going to its help and clicking (or typing RET) on its source file name.

Notice how `M-SPC` is implemented in terms of a more-general function, which does not have a keybinding. Use `M-:` to execute this more-general function on a buffer, such that the function changes the buffer's contents.

Similarly, use `M-x` to execute the more-general function on a buffer.

Exercise 2.2: Scripting Emacs

Use the Emacs command `M-x what-line` and see what it does.

`M-x what-line` simply tells you what line you are on, not how many lines are in the buffer. Design and implement a command `M-x which-line` that acts like `M-x what-line` except that it says "`Line 27 of 106`" in contexts where `M-x what-line` would merely say "`Line 27`". Do this by using `C-h f` to get help about `what-line`, navigating through that help to find its source code, putting a copy of the source code into a new file `which-line.el`, editing that file, loading it into Emacs, and then executing your new command.

When counting all the lines in a buffer, simply count the number of newline characters that it contains. This means that if a buffer does not end in a newline, you should not count the characters after the last newline to be part of another line. In particular, an empty buffer has zero lines.

Homework: Python scripting

- The Python Foundation, The Python Tutorial (2020)

Consider the old-fashioned Python 2 script `randline.py`.

What happens when this script is invoked on an empty file like `/dev/null`, and why?

What happens when this script is invoked with Python 3 rather than Python 2, and why? (You can run Python 3 on the SEASnet hosts by using the command `python3` instead of `python`.)

Use Emacs to write a new script `shuf.py` in the style of `randline.py` but using Python 3 instead. Your script should implement the GNU `shuf` command that is part of GNU Coreutils. GNU `shuf` is written in C, whereas you want a Python implementation so that you can more easily add new features to it.

Your program should support the following `shuf` options, with the same behavior as GNU `shuf`: `--echo` (`-e`), `--input-range` (`-i`), `--head-count` (`-n`), `--repeat` (`-r`), and `--help`. As with GNU `shuf`, if `--repeat` (`-r`) is used without `--head-count` (`-n`), your program should run forever. Your program should also support zero non-option arguments or a single non-option argument "`-`" (either of which means read from standard input), or a single non-option argument other than "`-`" (which specifies the input file name). Your program need not support the other options of GNU `shuf`. As with GNU `shuf`, your program should report an error if given invalid arguments.

Your solution should use the `argparse` module instead of the obsolescent `optparse`. It should not import any modules other than `argparse`, `string` and the non-`optparse` modules that `randline.py` already imports. Don't forget to change its usage message to accurately describe the modified behavior.

What happens when your `shuf.py` script is invoked with Python 2 rather than Python 3, and why?

Submit

Submit the following files within a compressed tarball named `assign2.tgz`.

- `which-line.el`
- `shuf.py`
- `notes.txt`, a text file answering questions and containing any other notes or comments that you'd like us to see.

All files other than the `.drib` files should use GNU/Linux style, i.e., UTF-8 encoding with LF-terminated lines.

The shell command:

```
tar -tvf assign2.tgz
```

should output a list of file names that contains `which-line.el` etc., with sizes and other metainformation about the files.

$Id: assign2.html,v 1.3 2020/10/15 22:16:02 eggert Exp $