Week 11 Homework 💻 🐖

New stuff learned this week:

Web Servers:

- a web server is a computer *program* that *listens* on a *port* for inbound HTTP *requests* and returns HTTP *responses*.
- `nginx` is the most popular current web-server program, pronounced "engine X"
- a really simple, and fast strategy for web servers is to respond to HTTP `GET` requests by trying to hand back *premade HTML files*. This is how I have our server set up currently, and it's what the `try files` line in the `nginx` config means.
- web servers can also make up new HTML on the fly when responding to a request — this is *slower* but more *powerful*
- I configured `nginx` to have your *subdomains* **WEB ROOT** to be the `~/www/` folder — that's where `nginx` will look when it is *trying to find files to match requests*

URLs:

- a `URL` stands for "**U**niform **R**esource **L**ocator", but it's easier to think of it as an *address* on the internet to look for something, an example is `http://google.com/cats`
- the `http://` or `https://` portion of a URL is called a **SCHEME**
- something like `google.com` or `howtocomputer.link` or `harvard.edu` is called just a **DOMAIN**, or sometimes a **REGISTERABLE DOMAIN** (because that's what normal people can buy, or register as their own)
- the `.com` or `.link` or `.edu` or `.co.uk` part that comes at the *very end* of a domain is called the **TOP-LEVEL DOMAIN (TLD)**
- anything that comes between the *scheme* and the *domain* is called a subdomain: `http://<subdomain>.google.com` — in fact if you've ever been to a site that started with `www` like `http://www.duolingo.com` — the `www` part is technically a *subdomain*.

Vim:

- 4 common *operators* are:
  - `d` - delete
  - `c` - change (delete then go automatically into insert mode)
  - `y` - yank (copy)
  - `v` - visual (select visually, or highlight)
- a few really useful *text objects* are:
  - `w` - word

    - `p` - paragraph
    - `t` - HTML tag
- two special (and super useful!!) quantifiers for text objects are:
    - `i` - inner (matches a whole text object, no matter where you are in it)
    - `a` - around (matches a whole text object + a space or blank line, no matter where you are in it)
- you can *combine* operators, motions, and text objects in a very intuitive and extremely powerful way, here are some examples:
    - `ciw` - **C**hange **I**nner **W**ord
    - `daw` - **D**elete **A**round **W**ord
    - `yap` - **Y**ank **A**round **P**aragraph
    - `dit` - **D**elete **I**nside (HTML) **T**ag
    - `vip` - **V**isually-select **I**nner **P**aragraph
- the `f` and `t` operators let you *jump to, or right before* a character
    - so `fx` would jump you *onto* the first `x` in the line
    - and `tx` would jump you *right before* the first `x` in the line
- you can use `f` and `t` with the operators too, like so:
    - `dfx` - **D**elete through the first **F**ound **X** character
    - `ctx` - **C**hange up **T**o the first **X** character
- if you've done something like `ciw` and then typed a word and gone back into regular mode, the `.` character REPEATS the last edit you made 🍾

Regular Expressions:

- the `\d` special symbol means "match any ***d***igit" — it's exactly the same as `[0-9]`
- if you want to specify *exactly how many* of something you want to match, you can do that with a special quantifier `{<number>}` like `o{3}` will match only if there are *exactly* 3 `o` characters in a row
- if you want to specify a *range* of numbers, you can do so with two numbers, separated by a comma, like `0{3,5}` which will match between 3 and 5 consecutive `0` characters
- if you want to express something like *3 or MORE* matches, you can use the comma but leave off the last number, like `0{3,}` — that would be the same as like writing `0{3,9999999999}`
- the `{}` quantifier operates not only on individual characters, but on whatever the preceding *token* is, which can be a *character, a character class, a parens group,* etc., like these examples:
    - `f{3}`
    - `[a-f]{2,7}`
    - `(foo|bar){5}`

HTML:

- the `<ul></ul>` html *tag* creates an **U**nordered **L**ist
- the `<ol></ol>` html *tag* creates an **O**rdered **L**ist
- both list types have *children* of `<li></li>` — **L**ist **I**tems
- example: `<ul><li>Item 1</li><li>Item 2</li></ul>` (but would be better to put each *tag* on a *new line*, which I can't do in this Slack post)

Touch Typing Links:

- http://touchtype.co
- https://www.how-to-type.com

Homework plan:

A little lighter this week because of Thanksgiving 🦃

- 1 day reviewing and creating a few more flash cards
- 1 days CLI/Regex practice
- 1 day `vim` practice
- 1 day touch-typing practice
- 1 day ***WEB*** practice

Homework day 1:

- do flashcard assignment (see below)
- touch typing practice

Homework day 2:

- `vimtutor` - Everything except Lesson 7 (but USE your new `vim` skillz)
- CLI practice

Homework day 3:

- Web Practice

Flash Card Assignment

- Review all of your old cards
- Make a new `REGEX` card covering `{3}` and `{2,5}` quantifiers
- Make 9 new `VIM` cards covering:
  - `f<char>`
  - `t<char>`
  - `ciw`
  - `caw`
  - `cit`
  - `yap`
  - `viw`

    - `daw`
    - `.`

CLI Homework:

⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️
IMPORTANT NOTE: `sed` doesn't support the `\d` character, so instead of using `sed -E` I want you to use `perl -pe` — `perl` is another program that can work like `sed` and it does support `\d` — the rest of the syntax is *exactly the same:* `perl -pe 's/foo/bar/gi'` is exactly the same as `sed -E 's/foo/bar/gi'`
⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️

1. slowly and carefully review the "Regular Expression" portion of the "New stuff learned this week" above ^^^.
2. `ssh` into your home dir and make a `week11/` directory
3. copy the `numbers.txt` and `letters.txt` files from a folder called `regex` which is inside the computers *root dir* into your `week11/` dir
4. `cat` out the `numbers.txt` file and then use `perl` (and the `\d` token, plus the other new stuff learned this week) to make it so line 1 reads `Here is my phone number #secret#`
5. change your regular expression so that it also changes the full phone number including area code on line 2 to `#secret#` - so that line 2 should now read `My landline is #secret#` — and the first line should still be the same as in step 3 above.
6. change your regex again so that it matches and replaces ALL the phone numbers on the first 5 lines with `#secret#`
7. **Extra Credit:** ✨ make a `perl` expression so that on the first 4 lines, all of the phone numbers are formatted easier to read like `(555) 111-2222` - but the numbers should be preserved (so, for instance, line 4 should read `Jenny's phone number is (555) 867-5309`)
8. make a new `perl` expression that matches *social security numbers (SSN)* — (one of the lines of the text explains how they are formatted) - replace the TWO valid social security numbers with `###-##-####` - but none of the phone numbers should be changed. 🤔
9. repeat step 9, but this time, also make it so that the `XXX-XX-XXXX` on line 9 is also changed to `###-##-####`
10. write a new `perl` expression that changes the three *year dates* on the second to last line with `#YEAR#` — but it should not change any of the other numbers in the whole file, including `5002` and `300` and the phone numbers, etc.
11. now, switch to `cat`ing out the `letters.txt` file, and write a regular expression with `perl` that changes line 1 to exactly `I l@ve y@mmy food` — notice how `food` is unchanged.
12. Next, change your expression so line 2 is changed to read `Jared Henderson likes jimjam.` — use an *empty replacement* `//` and make sure the line still ends with `jimjam.`
13. Finally, again using an empty replacement, write a regular expression so that the characters garbling up the middle of the word on the last line get removed, resulting in the last line reading: `I was born in Pontiac, MI.`

Web Homework

1. carefully and slowly review the HTML portion of *New stuff we learned this week* above ^^^
2. `ssh` into your home dir, and then `cd` into the `www` dir
3. you should still have `boilerplate.html` in that directory — `cat` it out one time to remind yourself what the different parts of a valid HTML page are
4. now, create a brand new file called `list.html` using `vim` and start by typing *FROM SCRATCH* a valid HTML file (including a doctype, html, head, title, body tags) — you can close vim and cat out the boilerplate a couple times if you need to refresh yourself. Give the new file a `<title>` of "My List" and a `<h1>` tag that says "Groceries to buy:". Save the file and view it in a browser.
5. edit the `list.html` file so that under the `<h1>` tag it contains an *unordered list* containing a minimum of 10 things to buy at the grocery store. — be sure to use your `vim` skills to do things like copy/paste lines `yyp` and `ciw` or `cit`. Save the file and view your `list.html` file in a browser.
6. quit out of vim, and using a shell command, *copy* the `list.html` into a new file called `vim.html` — then open that file in `vim`
7. inside of `vim` - change the `<title>` tag and the `<h1>` so that they both read `Steps to master vim`
8. then, change the *unordered* list into an *ordered* list.
9. now, try, with ONE command in `vim` to remove all of the `<li>` tags
10. make at least 5 new `<li>` tags describing steps to master vim
11. save the file and view it in a browser — compare it to your `list.html` webpage — how do browsers render (draw on screen) the difference between an *unordered* and an *ordered* list?
12. at the bottom of each of the two files you made in this homework session, add a link from one to the other, so your `list.html` file should have a clickable bit of text that says `Check out my steps to master vim!` that links over to the `vim.html` file, and your `vim.html` file should have a link that reads `Here's my grocery list` — and they should both be clickable and work in a browser to navigate back and forth between the two pages. (reminder a link looks like this: `<a href="<URL>">Some text</a>` where `<URL>` is the thing you're linking *TO*)

- **Extra credit** ✨ add to one of your web pages a few links that go to other students' vim and list html pages.