Software Carpentry Bootcamp -- Day 2

http://swcarpentry.github.io/2013-08-23-harvard/

Twitter: @swcarpentry #dst4l
Class-related URLs: https://pinboard.in/u:dst4l

Python III

Mike Selik
http://swcarpentry.github.io/2013-08-23-harvard/lessons/swc-python/word-frequency.html
Detecting spam
Grammar, syntax, keywords (i.e. Viagra), domain name, quantity vs variety
Frequency of words is common method

You can usually try googling what you want to do, and if it's common, there's probably a python library that can help. For example, searching "Python download file from web" can lead to http://docs.python-requests.org/en/latest/, a library for you to install.

Reuse, recombine pieces of code to create new code.

When you're using a library that's not one of the defaults, you have to import it with an import statement. Before you import a library (such as requests), commands related to that library will not be meaningful to python.

You import what you need when you need it. This can help avoid naming conflicts between libraries and variables.

"Explicit is better than implicit."

Don't name a variable "str" -- the function "str" will be unusable. Variables can be deleted, but there is no way to clear them all at once. In ipython, %whos will show you all variables.

List: a data type used to store a bunch of data in a particular sequence

- range(): creates a sequence of numbers as a list
  - range(9) = [0, 1, 2, 3, 4, 5, 6, 7, 8]
  - range(1,10) = [1, 2, 3, 4, 5, 6, 7, 8, 9]
  - range(5, 10, 2) = [5, 7, 9]
- Python is what's called a "zero indexed" language. That means when counting objects in a collection we start counting from "zero" instead of "one" For example the first element of a list is the 0th element. The first character in a string is the 0th character. For example:
  - >>> my_str = 'Hello'
  - >>> my_str[0]
  - 'H'

    - >>> my_list = [1, 2, 3]
    - >>> m_list[0]
    - 1
    - >>> my_list[1]
    - 2
- negative index numbers: count from the end of the list; my_list[-1] gets the last element in my_list
- Slicing a list: a subset of a list, in sequence, between index numbers:
    - list[0:3] gets the first three elements in list
- Strings are also indexed, so they can be sliced like a list. A list is a container and can be changed, appended.
    - You can use the ".append()" function on a list, as it is not immutable, although it has to be stored as a variable for you to print it out. You can't append to strings.

- Parentheses indicate two things: a tuple, which is an immutable list, and calling a function with parameters (which apparently are also a tuple?).
    - my_tuple = (1,2,3,4,5) declares a tuple with the name my_tuple
    - my_function(1,2,3,4,5) runs the function my_function with parameters 1,2,3,4,5

- **Square brackets [ ]** indicate that the contents are a list, or we're pulling something out of a list when at the end of a variable, so look for my_list[2].
- **Parentheses ( )** indicate that we're calling a function

\r and \n are artifacts of typewriter interaction: separate carriage return and new line commands

u'text' indicates unicode characters

- **splitlines()** is a function that takes no arguments. len() takes one argument, and range() takes two arguments, and all of the arguments go into the parentheses, and if there aren't any, the parentheses are still there.
- **split()** splits on specified characters:
    - split('\n')
    - default splits on spaces, tabs, returns
    - it's built in to python, and relatively simple, but if you want to do more sophisticated analysis, you can use a more sophisticated library.

Loops:

basic loop syntax:
for *some variable* **in** *something***:**
    do stuff with *some variable*
It's a best practice to call *some variable* something that relates to what you're iterating over. And in general, name things so that others can understand your code, and also so that you can understand the

code when you go back to it.

python understands the word after "for" as a variable, no matter what it's called.

Dictionaries:

"dict" for short

in curly brackets with colons { 'key':value, 'key':value}. values can be anything, and are indexed by keys, not by whole values. Lists can be stored in dictionaries.

create an empty dictionary by assigning dict() as a variable value

to create a paired key and value in a dictionary:
dict_name['key'] = value

Python IV

exercises: http://www.codecademy.com/courses/python-beginner-en-kSQwt

format() uses {} but does not involve dicts, the brackets just indicate very temporary variables.
"I'm a {0}! Look at me {1}!".format('cheetah', 'run') will return "I'm a cheetah! Look at me run!"

tip: control L clears screen in just about any terminal

deque (http://docs.python.org/release/2.5.2/lib/deque-objects.html)

an alternate way to concatenate strings (as opposed to using +):

```
names = ['Adam', 'Tanya', 'Alice', 'Sean']
for friend in names:
    print "Hello {0}".format(friend)
```

It's technically faster for python to compute the format command.
You can use multiple values {0}, {1}, etc.; the number is {} points to an element in the tuple ()

- .startswith() returns true or false based on whether the argument is in the string that you apply this to
  - use in lieu of string[0:?] == value
  - only works with strings

```
words = ['zip', 'blip', 'croissant', 'toast', 'quip', 'quail', 'quetzal', 'quizzical']
for word in words:
    if word.startswith("q"):
        print word
```

can also use .endswith
check out string documentation for others, like .lower, .upper, etc.
All of these neat things are functions, and you can find out what functions are available for a particular object by searching for something liks "python string functions" or python list functions," and that will probably link you to relevant documentation.

conditions can be combined within an if statement using "and", "or"

```
words = ["pizazz", "python", "zebra", "pizza"]
for word in words:
    if word.startswith("z") or word.endswith("z"):
        print word
```

Python uses tab indents to show whether you're in or outside of an if statement. Other programming languages cruelly make you use brackets, etc. ("There's even this one language called LISP that is, like, all brackets.")

using for loops with lists:

```
honor_roll_count = 0
student_grades = ["A", "C", "B", "B", "C", "A", "F", "B", "B", "B", "C", "A"]
for grade in student_grades:
    if grade in "AB":
        honor_roll_count += 1
print honor_roll_count
```

Commenting

- comments are for things that the code doesn't tell you. Anything following a # is not read by the program.
- comments are one line.
- to comment multiple lines, can put a # in front of each line. You may also see 3 quotation marks surrounding a multi-line comment, especially to explain functions.
- many code editors and IDEs ("integrated development environments") will comment out multiple lines of code, e.g., Komodo Edit

"Programmers are very fervent about their favorite text editors."

You may want to intentionally place a print() command within a loop when testing your code, so that you can monitor variable values as your loop is executed.

- You can quickly create a list of letters from a string:
  - [letter for letter in string] returns [first letter, second letter, third letter...]
- .join() will create a string by joining the elements from a list

Version Control I

http://nicercode.github.io/blog/2013-04-05-why-nice-code/

http://www.phdcomics.com/comics/archive.php?comicid=1531

Note:For anyone at Harvard (or others with personal or institutional access), lynda.com has a tutorial on Git

Why use version control?

Undo changes
Access snapshots of a project consisting of multiple files at various points in time
Make collaboration easier: know who edited what and why; submit proposals for changes before implementing; edit the same version of the file(s)
Go find some specific edit

Track revision history for any type of file: source code, prose, resume!
Viewing diffs is less helpful in images or Word docs. There are tools that make it useful, but they usually aren't free.

Fork: copy someone else's work and its history to your own repository; make your own changes without affecting original author's repository

Git: One of many version control systems. Git was designed by programmers for programmers. Popular.

- **$ mkdir** creates new directory
- **$ git init** creates an empty git repository in directory
- just need to do this the first time you set up your repository:
  - **$ git config --global user.name "Your Name"**
  - **$ git config --global user.email "eslao@fas.harvard.edu"**
  - you can check this by typing cat ~/.gitconfig, and you should see your name and email
- **$ git status** shows tracked and untracked files

- **$ git add filename** stages file before committing
- **$ git commit -m "useful commit message"** commits current versions of tracked files and records history

"commit"
verb: take snapshot of repository
noun: the snapshot that you take of a repository

Don't be like this: http://whatthecommit.com/30e0a72af9d239226aeaadaac540e8ad

Commit messages can be as long as they need to be.

Why do you need to **$ git add** every time you want to commit?
"It's kind of a safety valve."
"You may have changed multiple files, and those changes might be unrelated."
Each commit should have a single purpose.
With each change being a single purpose, you can roll-back one change but not the other.

Process is:
$ git add [filename]
$ git commit -m "useful commit message - useful description"
(make changes)
$ git add filename
$ git commit -m "useful commit message - useful description"
(make changes...)

- **$ git log** shows commits, chronologically, from most recent
- **$ git log --help** git log takes lots of options
- **$ git diff** shows changes to files that have not been committed or staged ("add")
  - **--color** is a helpful option, shows red for deletions, green for additions.
  - **--cached** shows changes to staged files

Git will un-stage files if changes are made after add.

How to make Git ignore files:

create a text file called ".gitignore" and put in filenames of files that you want to ignore. These files can include wildcards, so you can ignore all files of some type. It's a good idea to ignore files that include logins or other sensitive information.

"Git" is the version control system
"GitHub" is a commercial service for sharing repositories online (free for public accounts, pay for private accounts)

.md files use markdown: http://daringfireball.net/projects/markdown/syntax

Demo of using git for managing a thesaurus/taxonomy
https://github.com/selik/thesaurus-demo

- **$ git clone https://url.here** to clone git repository from a web service
- **$ git push** to send files and changes to online repo; will ask for login by default

GitHub allows some online editing, conversations about pull requests, and comments on specific diffs.

A Github tutorial is located at http://try.github.io/

Check out https://github.com/selik/thesaurus-demo for an example of how to collaboratively manage a thesaurus using GitHub.

Also take a look at https://github.com/selik/harhamhuckle, which will contain a correct and expanded example of the word-frequency exercise that you can use as a starting place for your future text analysis scripts.

Regular Expressions

A pattern used to find matches in text
great for finding things, finding and replacing things
works on text only

(Downloaded http://bit.ly/allwords to desktop as words.txt - it's a file containing every valid Scrabble word)

- **$ wc filename.txt -l** counts lines in a file
- **$ grep "string" filename.txt** finds lines that contain a string
- **$ grep "^string" filename.txt** finds lines that start with the specified string
- **$ grep "OO$" filename.txt** finds lines that end with the specified string
- **$ grep "Q" words.txt | grep -v "QU"** looks for words containing Q and then searches those results for words that do not contain U
- **$ grep "^L.B..RY$" words.txt** finds 7-letter words that match this pattern
- **.** is one wildcard character
    - **$ grep "^A.Z$" words.txt** gets ADZ
- .* is any number of wildcard characters
    - **$ grep "^A.*Z$" words.txt** gets
        - **ABUZZ**
        - **ADZ**

- **ALFEREZ**
- **ASSEZ**

Ways to keep up/build Python skills:

"You never have to pay to learn to program."
Boston Python User Group: http://www.meetup.com/bostonpython/
Think Python: http://www.greenteapress.com/thinkpython/
Python Tutor: http://www.pythontutor.com/
Use the Harvard Software Carpentry email list to ask questions
Schedule an online consultation with Erik
Software Carpentry IRC channel: #swcarpentry on irc.freenode.net
If you're interested in web scraping: https://scraperwiki.com/