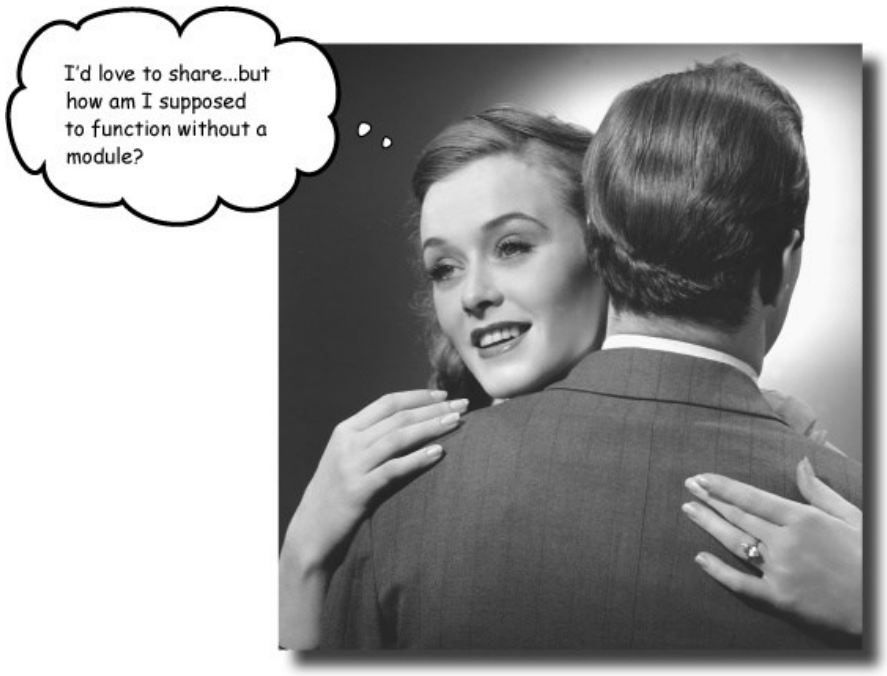

Searching for Terms on Webpages – B

Add regex searching and reporting to the websearch script

Reminder: the Task:

- Copy some webpages of interest, and save them into text files
  - They happen to be online news sites
- Search each webpage for specific terms
  - Multiple terms
  - Might be regular expressions, or constant phrases
  - Stored in a file
- Save a report of the matches between webpages and search terms in a file
  - File should be readable into a spreadsheet
  - ".csv" (comma-separated-values) format is human-readable, convenient

Easy Searching

- Python can look for substrings within a string:
  - `mystring.find( substring )`
    - returns -1 if no match
  - `mystring.index( substring )`
    - throws exception if no match
  - `( substring in mystring )`
    - returns True or False, test with an **`if`** statement

Add searching to the webpage reader

- Get URL lists
- Open each URL in each list
- Read URL's contents
- Write contents to a local file
  - one local file per URL list

- Get search terms!
- Search for each term within each URL's contents

Two More Modules

- re
  - Regular expressions are powerful pattern matching tools
  - We'll include some options
- csv
  - Write data in a format that makes spreadsheets happy
  - Also human-readable, unlike .xls, .xlsx formats

xkcd

Regular Expressions

- Patterns that match portions of text strings

```
taxes
(S|s)enator
shooting
accident(s)?/w
traffic
```

- Each character is an *atom*
  - Parenthesized sub-expressions act like atoms
- Special symbols
  - **|** indicates exclusive *OR* – match either character
  - ***?*** indicates count of preceding atom – 0 or 1
- Character class **\s** represents any whitespace

Regular Expressions – Some Features

- Supplied: list of search terms
  - Terms can be regular expressions
- Desired: search term *plus* 150 preceding characters, 2000 following characters of "context"
  - ...if there are that many characters, before or after
  - Context isn't all on the same line as the search term – spans multiple lines of source text
- Characters may not all be simple ASCII
- Text may contain multiple matches to term
- Matches may not be the proper case

Matching Context

- Full regular expression:
  - Up to 150 characters (any character)
  - Provided search term
  - Up to 2000 characters (any character)
- Pattern:
  - `.{,150}term.{,2000}`
- Dots must match *any* character, including newline characters
  - Special flag sets this behavior: `re.DOTALL`

Performance and Efficiency

- Pattern is large, may take a long time to work
- Additional special flags needed
  - `re.IGNORECASE` – match uppercase or lowercase
  - `re.UNICODE` – match even if non-ASCII characters
- Pre-compile the pattern for better performance

```
regex = re.compile(          \
    '.{,150}term.{,2000}',   \
    flags=re.DOTALL          \
    | re.IGNORECASE          \
    | re.UNICODE             \
)
```

Multiple Matches

- Various match-finding methods:
- **re.match()** – most restrictive, requires complete match from beginning to end of text
- **re.search()** – finds 1st match within text
  - Longest possible match, if variable length is possible
- **re.findall()** – finds multiple instances of matches within text
  - matches are *non-overlapping*
- **re.finditer()** – like **re.findall()** but returns iterator instead of list
  - More efficient if many matches

Exercise

- Download "https://montcs.bloomu.edu/Readings/zombies.txt"
- Form regular expression that matches "the" as part of another word, but *not* as a word by itself
- Try re.search(), re.findall(), re.finditer()
- Add context
- Try re.findall() again, with the context

(What About Overlapping Matches?)

- The commonly used re module does not recognize multiple matches that overlap each other
  - '((work)?book(mark)?)' matches:
    - "book", "workbook", "bookmark", *or* "workbookmark"
- So
  ```
  re.findall( '(work)?book(mark)?', \
  "What does workbookmark mean?" )
  ```
  won't find all matches
- The new "regex" module handles this
  - Non-standard, must be added to distribution
  - We don't have it 

Spreadsheet Files

- Various formats
  - .xls, .xlsx – MS Excel format
  - .ods – Open Document Format
  - .csv – Comma-Separated-Values
  - others as well
- Binary versus human-readable formats
  - .xls, xlsx, .ods formats preserve formatting, other features
    - Generally only usable w/ spreadsheet program
  - .csv format only saves cell contents (and cell structure)
    - Readable and usable by anything

Python and Spreadsheets

- **xlrd** module supports .xls, .xlsx formats
  - Non-standard
- **odfpy**, **ezodf** modules support .ods, .odf, etc.
  - Also non-standard
- **csv** module supports .csv files
  - Standard
  - Portable to all spreadsheet programs
  - No formatting; cell contents only
- Spreadsheet rows correspond to Python lists
  - So full spreadsheet corresponds to a list-of-lists
    - LOL ???

csv Module

- File must be opened for reading or writing first
- Opened file handle is supplied to
  csv.reader() – returns lists of rows
  or
  csv.writer() – writes rows to file
  - Value separator can be specified
    - Defaults to a comma
- For writing out:
  - csv.writer() object provides ".writerow()" method to write out a list as a spreadsheet row

Exercise – writing .csv

- Build 10 lists of 20 random values
  - np.random.randint()
- Write them into a .csv file
- Inspect file with text editor
- Inspect with spreadsheet program
- New python code:
  - Read each row from .csv file into list
- Calculate arithmetic mean, geometric mean, harmonic mean

Exercise- reading .csv

- Browse to https://www.eia.gov/state/ (https://www.eia.gov/state/)
  - Download "download table data as csv"
- Open file with "csv.DictReader()"
  - Read in rows of dictionaries
- Form lists for:
  - state abbrev.
  - production
  - consumption
  - expenditure
- Make bar graphs