CSE250, Spring 2022   Assignment 2   Due Wed. Mar. 2, 11:59pm

Lectures and Reading:

Chapter 6 is the place where the "Scala features..." part of the course transitions to the subject of data structures in general, which is the main content of the course. Focus on sections 6.2 and 6.3. Skim section 6.4 on "Buffers"—this section is mainly the laundry-list of operations on page 199, a few of which were covered in the example during Chapter 2 involving prepending and appending with arrays and lists. (Regarding that example and the timing analysis that followed, note the opening paragraph of section 6.1.1, where the text says: "This means that 'updates' can be made without making complete copies...the new object will share as much memory as possible with the original." This is what my Week 2 notes diagrammed.) By Monday we will already be into Chapter 7, which segues into the first numerical content of this course, O-notation. To supplement the text's short coverage of that, please read—really view—over the weekend these materials in the "Other Resources" section of the course webpage:

- https://cse.buffalo.edu/˜regan/cse250/order-notationhandout.pdf (a 2-page quick reference)
- http://science.slc.edu/˜jmarshall/courses/2002/spring/cs50/BigO/index.html
- https://slideplayer.com/slide/2811606/ (updated link for Grunschlag slides).

Then focus on sections 7.5–7.7, but skip the 'RPN' example in section 7.8. I tend to place "unit testing" into the vision of design by contract, emphasizing "requires" and "ensures" over spot-placed assertions, but we will need the latter to test the former anyway.

Then: *Chapters 8–11 are skipped.* Note that Chapter 7 introduces ADTs but the first chapter-scope example is Chapter 12. Suddenly by the end of next week we will be 2/3 of the way through this long text! But chapters 12, 13, 15, 16, 18, and 21–22 will be taken much more slowly, a week or more each for the meat of the course. Anyway, sections 12.1–12.3 (just 4 pages) include a conceptual recap of what was laid out in week 2 on lists and arrays. So please include them in the reading for Friday 3/4 and Monday 3/7, before the **First Prelim Exam** on Wednesday, 3/9.

——— *Assignment 2, due Mar. 2 "midnight stretchy" on CSE Autograder* ———

Brief Task Statement:

We will extend the Assignment 1 code to separate words and punctuation, and then consider when and whether hyphens should be regarded as part of a long word or as separating two shorter words. The first task is to write a method `tokenize` that converts each line of the input text file into a list whose strings are either a single non-alphanumeric character, or a *word* of alphanumeric characters (including the apostrophe). Show how using this method changes the answers on the Assignment 1 test files `Gettysburg.txt` and `JustHamlet.txt` away from the words "battle-field" and "comical-historical-pastoral" to something else.

Then write a method `dehyphenate` that works on each line separately and implements this policy, using a supplied dictionary file `wordsDWYL.txt` read as a set of words: If the hyphenated word is in the dictionary without the hyphen, like "battlefield," then remove the hyphen and count it as one word. Else, if the hyphenated parts are in the dictionary as separate words, leave them as separate words (while deleting the hyphen). Else, undo this part of what `tokenize` did by making it one long word with the hyphen(s) after all. Test whether your code works as-intended on the *triple*-hyphenated word "comical-historical-pastoral" even though your code only considers two hyphenated words at a time. (It is OK for your code to give dubious results when not all three words are in the dictionary. Note also that this is actually "tragical-comical-historical-pastoral" in *Hamlet* but the first word is hyphenated over the line in `JustHamlet.txt`. *It is AOK to ignore this*—don't try to combine lines. In fact, although `JustHamlet.txt` ends many lines with a double-dash, this is the only case of a word being hyphenated across a line. *You may ignore words hyphenated across lines in all tests, even if you get pieces that aren't words.*)

Thanks to coincidental feature of the dictionary as downloaded from (my `ScalaSamples` folder or) `https://github.com/dwyl/english-words` (it appears as "words.txt" but please give it the above name referencing the parent website `https://github.com/dwyl` for "Do What You Love"), your run on `JustHamlet.txt` will seem to have given a wrong answer by failing to remove an obvious hyphen. Before consulting the dictionary to see why, convince yourself that the bug does not happen because Shakespeare's hyphenated word appears first on an indented line. To fix the issue, make your storage of the dictionary non-case-sensitive. In an essay portion, say how much smaller this made your set object (if at all—and why if at all), after including the above discussion of how your `dehyphenate` method handles hyphenated chains of more than two words.

File(s) to Submit: One `.zip` file of the project root folder consisting of at least the following:

- `MaxWords.scala` (Yes, same file.) Location can be anwhere unique in the project but not in a `package`.
- A file `essay2.txt` in your project root folder, which must be the destination for input and output.
- A file `output.txt` of output from your program, on both `Gettysburg.txt` and `JustHamlet.txt`. (You can do separate runs rather than combine into one run, making sure you set the append option `true` when opening `output.txt` for output.)
- Optionally, other (small) text files you used for testing, again in the project root folder.

Do not submit your copy of `wordsDWYL.txt`; this will be supplied as a base file. See specific directions next. We will do other "hidden tests" of our own. The points are 30 for a working version of `tokenize`, 30 for `dehyphenate`, 30 for the whole code, and 18 for the essay answers, making **108 pts.** total. Manual grading with subdivided credit will be used, including some allowance for coding style and economy.

Specific Directions and Ground Rules (or Ground-Rules, or Groundrules)

- No pre-given code, but you may adopt some or all of the Assignment 1 answer-key code.
- Your first order of business should be to convert `matrix` from being type `Array[Array[String]]` to type `Array[List[String]]`. This should require only a few changes in declaration and use(s) of `.toList`. (It may seem strange to continue using array-style indexing with lists, but that's OK—especially since the lists are relatively short lines of text.)
- *Thou shalt not* use screen input for filenames, or for anything in a submitted file. (But screen output is OK—it will show up in the feedback.)
- The last line of your `output.txt` output must have the Assignment 1 form typified by

  ```
  The word of longest length 35 is List.tabulate(10)(_+1).zip(answers) in line 91, column 3
  ```

  (Of course, your answers themselves will change—depending on the stage of your code, it might be `Javascript-like` in line 85, column 15.)
- It is no longer OK to keep punctuation marks that touch your words and count them in the length. The apostrophe `'` counts as a letter character, so you should have `doesn't`, not `doesn :: ' :: t` in your lists. Hyphens should be separated by your tokenizer. E.g., `mustn't-do` is tokenized as `mustn't :: - :: do`, and then depending on whether `mustn't` is in the dictionary, left that way or joined into one word again by `dehyphenate`.
- OK if your `tokenize` itself does the entire file, or takes each line fed to it separately.
- You must enable a command-line argument for passing test files as on Assignment 1. This excludes the dictionary file.
- The dictionary must ultimately be read as `val dictFile = "../wordsDWYL.txt"` one level outside your project root folder. (In IntelliJ on Windows, this level is `.../Users/<you>/IdeaProjects/`.)
- The dictionary should be read in as a `Set[String]`; initially via the given code line `val dict = Source.fromFile(dictFile).getLines().toSet` but you will want to change this in the final steps of the project.
- Both `tokenize` and `dehyphenate` must use the `match ... case ...` mechanism of Scala. *At least one of them* must use the `::` operator "on the left-hand side"—that is, between `case` and `=>`. (In the other you may "punt" `match` to a corner of the code.)
- You may not use `java.util.StringTokenizer` or any third-party tokenizing code.
- Beware this **curveball**: The char `'-'` is not the same as the string `"-"`. Scala however allows trying to equate them, with only a warning that the answer is always `false`.

Essay Q1: Give at least two different examples of how your code handles triple-hyphenated words. One of them can instead be *how* it handles a word followed by a double-dash, like the end of lines in `JustHamlet.txt`. You should want to leave the word separate, but other behavior is OK if your essay calls attention to it here.

Essay Q2: How much does your `Set` of the dictionary words shrink (if at all) when you create it all lowercase? Can you make your code find out exactly why that happens?