


# <oo> → <dh> Digital humanities

Maintained by: David J. Birnbaum ([djbitt@gmail.com](mailto:djbitt@gmail.com)) 

Last modified: 2015-09-21T12:41:08+0000

## Regex assignment #2

### The task

Assume that we've been given a plain-text file like the Project Gutenberg EBook of [The Blithedale Romance](#), by Nathaniel Hawthorne, and we want to convert it to XML, but we don't want to type all of the angle brackets manually. (Note that this site sometimes shows you a pop-up welcome screen and a list of different versions of the file, instead of taking you to the plain text one directly. If that happens, click "OK" on the "Welcome" pop-up and then select the version labeled "Plain Text UTF-8".) In this case Project Gutenberg makes the same book available in HTML, and in Real Life we'd probably convert from HTML to XML (using XSLT, which we'll learn later in the semester) rather than from plain text, but since there are situations where all we have is plain text, we'll ignore the HTML version on the Gutenberg site, pretend that all they provide is plain text, and work with that. So what can we tag automatically, with global find-and-replace operations? Some of the markup we might want to introduce for analytical purposes might require us to touch every word of the text, but, at a minimum, we can autotag chapters, chapter titles, paragraphs, and quotations using regex tools, and that's the goal of the present assignment.

### Preliminaries

Select the plain-text version of the document and open it in <Oxygen/> as a plain text file. Then cut out the front matter (before the main title title) and the back matter (after the last line of the text of the novel, which is "I--I myself--was in love--with--Priscilla! "). In Real Life we might want to mark those parts up eventually and reintroduce them into the XML as metadata, but for this assignment we'll just delete everything that isn't part of the text of the novel.

### Step by step

There's more than one way to accomplish this task, but one way to approach the problem is as follows:

### Reserved characters

The plain text file could, at least in principle, contain characters that have special meaning in XML: the ampersand and the angle brackets. You need to search for those and replace them with their corresponding XML entities; if you don't remember the entity strings, you can look them up in the "Entities and numerical character references" section of <http://dh.obdurodon.org/what-is-xml.xhtml>. Note that you need to process them in the correct order. What is that order, and why is it important?

### Extra blank lines

The blank lines are pseudo-markup that tell us where titles and paragraphs begin and end, but in some cases there are multiple blank lines in a row (for example, there are two blank lines between the title and the word “by”). Those extra blank lines don’t tell us anything useful, so we’ll start by getting rid of them. We want to retain one blank line between titles and paragraphs, etc., but not more than one.

To perform regex searching, you need to check the box labeled “Regular expression” at the bottom of the <oXygen/> find-and-replace dialog box, which you open with Control-f (Windows) or Command-f (Mac). If you don’t check the “Regular expression” box, <oXygen/> will just search for what you type literally, and it won’t recognize that some characters in regex have special meaning. You don’t have to check anything else yet.

The regex escape code that matches a new line is `\n`, so you want to search for more than two of those in succession, and you want to replace them with exactly two. You can search for three blank lines and replace them with two and then keep repeating the process until there are no instances of three blank lines left, or, more elegantly and efficiently, you can search for `\n{3,}`, which matches three or more new line characters in succession (see the “Limiting repetition” section of <http://www.regular-expressions.info/repeat.html>) and replace them with `\n\n` (the quantifiers work only in matches, but not in replacements, so you have to write it this way).

Note that a transformation that searches for a sequence of two end-of-line characters depends on their being immediately adjacent to each other. If what looks like a blank line to you actually has (invisible) spaces or tabs, the pattern won’t match and the replacement won’t happen because there will be spaces or tabs between the end-of-line characters, which is to say that they won’t be adjacent. If you think that might be the case, you can make those characters visible by going into the <oXygen/> preferences (Preferences → Editor) and checking the boxes labeled “Show TAB/NBSP/EOL/EOF marks” and “Show SPACE marks” under Whitespaces. If you do have whitespace characters interfering with your ability to find a blank line (that is, two consecutive new line characters), you can use regex processing to replace them: the pattern `\t` matches a tab character, a space matches a space, and `\s+` matches one or more white-space characters of any sort (including new lines). You can use the “Find” or “Find all” options in the find-and-replace dialog to explore the document and make sure that you’re matching what you want to match before you use “Replace all” to make the changes.

## Paragraphs

What’s left after deleting the beginning and ending metadata and extra blank lines is mostly (except for the stuff at the top) a bunch of chapter titles and paragraphs, separated from one another by a single blank line, and we can use a regex to find all blank lines and replace them with the sequence `</p><p>`. XML doesn’t care about the following, but for human legibility, we’d suggest inserting a new line character between the tags, instead of just outputting the end tag followed immediately by the start tag, so that each paragraph will start on a new line. You’ll have to add the `<p>` start tag before the first paragraph and the `</p>` end tag after the last one manually, but you can enter all of the rest automatically with a single regex-aware find-and-replace operation. At this point the document looks like a bunch of `<p>` elements. Some may contain chapter titles, rather than paragraphs. We’ll fix that below. At the top of the file, the title, author, and list of chapter titles will need special handling. We’ll talk about those below, too.

## Chapter titles

The title of the first chapter within the body looks like:

```
<p>I. OLD MOODIE</p>
```

the second looks like:

```
<p>II. BLITHEDALE</p>
```

and we can see easily, from the list of chapter titles at the top, that there are twenty-nine chapter titles, each of which begins with a Roman numeral, then a period, and then a single space character, and each of which runs until the end of the line. No real textual paragraph looks like that, although some paragraphs could begin with the pronoun “I”, which looks like a Roman numeral, and some paragraphs might be only one line long. If we can write a regex that matches chapter titles and only chapter titles, then, we can replace the paragraph markup with title markup, retaining the part in the middle.

We’re not going to write that regex for you, but we will tell you the pieces we used. Try building a regex and running “Find all” to verify that it is matching all of the chapter titles and nothing else. When you can match what you need, then you can think about how to craft the replacement string. Here are the pieces:

- First make sure that, under “Options”, “Case sensitive” is checked and “Dot matches all” is unchecked. You want to do case sensitive matching because the Roman numeral characters here are all upper case, so you want to be able to distinguish those from lower case “i”, “v”, “x”, etc. We’ll discuss when to use “Dot matches all” below, but for now, make sure that it’s unchecked.
- You want to match the entire content of a line, and you can do that by using the **^** (line start) and **\$** (line end) *anchor* metacharacters. If you type, say, “A” into the “Find” box and hit “Find all”, you’ll match every upper-case “A” in the document (try it). But if you type “^A”, you’ll only find an “A” at the very beginning of a line. In other words **^** doesn’t match a caret character; what it does is anchor the match so that it succeeds only if it falls at the beginning of a line. Similarly, the **\$** doesn’t match a literal dollar sign; what it does instead is anchor a match so that it succeeds only if it falls at the end of a line. This also means that **^A\$** will match only lines that consist of nothing except the letter “A”. You don’t want to match lines that consist of nothing but the letter “A”, but with the two anchors (**^**, **\$**), the knowledge that the dot (**.**) matches any character except a new line, and your knowledge of the regex repetition indicators (**?**, **+**, **\***), you have all the pieces you need to craft a regular expression that will match an entire line, no matter what the contents.
- A chapter title is (now) wrapped (misleadingly) in **<p>** tags and fills a single line. That may not always be the case with other texts you’ll need to process, but you can see that it is here, and you can take advantage of that fact by searching for lines that begin with **<p>** and end with **</p>** (using the line start and line end metacharacters to match those tags only at the beginning and end of lines). But you also need to match the stuff between the tags, and that’s different for every chapter. You can handle that situation, as you did when you matched entire lines above, by using the regex dot (**.**) metacharacter, which matches any character except a new line. And since you don’t know the exact number of characters in each title, you can match one or more characters by using the plus sign (**+**) *repetition indicator*, which means “one or more”. Now try putting these pieces together and matching all lines that begin with a **<p>** start tag, continue with one or more characters (any characters except a new line), and end with a **</p>** end tag. When you look at the results, you’ll see that you’ve matched all of the chapter titles, but also all other one-line paragraphs. You’ve also matched the title, author, and a few other lines near that top, but you’ll need to repair those manually at the end anyway.
- You now need to refine your regex so that you’ll continue to match chapter titles, but not other one-line paragraphs. Since chapter titles begin with a Roman numeral, you can modify your regex to match only if a Roman numeral immediately follows the **<p>** start tag. To do that you’ll use a *character class*, which you can read about at <http://www.regular-expressions.info/charclass.html>. You want to match any sequence of “I”, “V”, and “X” characters in any order. This will match

sequences that aren't Roman numerals, like "XVX", but those don't occur, so you don't have to worry about them. This illustrates a useful strategy: a simple regex that overgeneralizes vacuously may be more useful than a complex one that avoids matching things that won't occur anyway. You can use the character class (wrapped in square brackets) followed by a plus sign (meaning "one or more") to enhance your regex and match only one-line paragraphs that begin with something that looks like a Roman numeral. Try it.

- This almost works, but it also matches one-line paragraphs that begin with the first person singular pronoun "I", such as:

```
<p>I mentioned those rumors to Hollingsworth in a playful way.</p>
```

To weed those out, you want to match a Roman numeral only if it's followed immediately by a period. Since the dot in regex is a metacharacter that matches any character except a new line, if you want to match a literal period, you have to *escape* the dot character by preceding it with a backslash (\). Add that after the Roman numeral part of your regex, and you should be matching only the twenty-nine chapter-title lines.

- Matching the chapter titles is necessary but not sufficient: you now need to replace the paragraph tags with **<title>** tags. To do that we need to *capture* the part of the title line that's between the paragraph tags and write that captured text into the replacement. To capture part of a regex, you wrap it in parentheses; this doesn't match parenthesis characters, but it does make the part of the regex that's between the parentheses available for reuse in the replacement string. For example, **a(b)c** would match the sequence "abc" and capture the "b" in the middle, so that it could be written into the replacement. Capturing a single literal character value isn't very useful because you could have just written the "b" into the replacement literally, but you can also capture wildcard matches. For example, **a(. )c** matches a sequence of a literal "a" character followed by any single character except a new line followed by a literal "c" character. You can use that information to capture everything between the paragraph tags in the matched string. To write a captured pattern into the replacement, use a backslash followed by a digit, where **\1** means the first capture group, **\2** means the second, etc. In this case you're capturing only one group, so you can build a replacement string that starts with **<title>**, ends with **</title>**, and puts **\1** between them. You don't need to do anything about the line start and line end anchors; since you've matched an entire line, the replacement will automatically be an entire line.
- Putting this all together, you should be able to retag your titles automatically, distinguishing them from the paragraphs. Try it.

## Chapters

A book isn't just a series of paragraphs with titles strewn among them; the book has logical chapters, which must begin with a title, and you want to represent this part of the logical document hierarchy in your markup by inserting **<chapter>** tags. Much as you used blank lines as *milestone* delimiters between paragraphs earlier, you can now use your **<title>** elements as delimiters between chapters. Use a find-and-replace operation to do this; you'll have to clean up the markup before the first chapter and after the last one manually, since in those cases the **<title>** element doesn't have the same milestone function as elsewhere.

## Quotes

How are quotations represented in the plain text? How would you find the text of a quotation, that is, how would you find where it starts, where it ends, and what goes between the start and the end? Files on the

Internet sometimes have errors and inconsistencies; if you're relying on cues in the text to identify the beginnings and ends of quotations, what can happen if you miss one?

If we assume that a quotation is text between opening and closing quotation marks (which are the same in this text, which has straight quotation marks, instead of the curly typographic ones where the opening and closing shapes are different), we have at least two concerns:

- A line may have more than one quotation. If we write a regex like `" .+ "` (including the quotation marks), will we match each quotation individually, or will we match the first quotation mark on the line and the last, erroneously gobbling up everything between into one spurious quotation? Try it and see.
- Some quotations span multiple lines. Since the dot matches any character except a new line, if we write `" .+ "` and the start and end quotation marks are on different lines, we'll fail to match those quotations, and we may erroneously match material between ending and starting quotation marks, instead of only between starting and ending ones. Try it and see.

Let's address the second problem first. There's a line in the text that reads:

```
without further question, only," added she, "it would be a convenience
```

which represents the end of one quotation and the beginning of another. If we write `" .+ "`, the system will incorrectly think that the first quotation mark opens a quotation and the second closes one, and it will also fail to recognize that the material before and after that line is really part of a quotation. We can fix this by checking the "Dot matches all" box, which changes the meaning of the dot metacharacter from "any character except a new line" to "any character including a new line". This means that we should be able to match quotations that cross line boundaries. Try it and notice the different results. Uh-oh!

So what went wrong? By default regular expressions are *greedy*, which means that they make the longest possible match. Turning on *dot all* mode causes the regex to match everything from the very first quotation mark in the entire text through the very last (since quotation mark characters are also characters, the dot in the regex `" .+ "` matches the quotation marks between the first and last ones in the document, just like it matches any other character). Turning off dot all mode won't fix this because some quotations do cross line boundaries, and we need to be able to recognize and match them.

We can resolve the problem by turning on dot-all mode (since we have to match quotations that span line breaks) but also specifying that the match should be *non-greedy*, that is, that we should make the *shortest* possible match (instead of the longest, which is the default), and we do this by following the repetition indicator (the plus sign) with a question mark. (Note that the question mark you met earlier is a repetition indicator that means "zero or one instance" of whatever it follows. Here it isn't a repetition indicator, though; here it means "don't be greedy". So if the same symbol can have two such different meanings, how does a regex processor know which meaning to apply?) In other words `" .+? "` will correctly treat two full quotations on the same line as separate quotations. Try it. You should now correctly be matching each quotation fully, regardless of whether it spans a new line character and regardless of the number of quotations on a line.

Once you can do that, you can capture the text between the quotation marks and write it into the output between `<quote>` tags. Don't include the quotation mark characters themselves in the capture group; those are plain-text pseudo-markup, and now that you're going to be tagging quotations with real markup, you don't want the quotation mark characters included.

## Cleanup



At this point you can fix the title and author lines manually (we'd just delete the line that reads "by", since the new `<author>` tags will make that explicit), as well as the table of contents, and you need to wrap the entire document in a root element (such as `<book>`). If you'd like a little more regex practice, instead of fixing the table of contents manually, you can use regex find-and-replace to tag it. If you select the table of contents and then open the find-and-replace dialog, you can check the "Only selected lines" box under "Scope" to say that instead of applying find-and-replace operations to the entire file, you'll apply them only to the selected lines. You may want to start by stripping out incorrect markup that you've inserted when your global find-and-replace operations earlier changed these lines, as well—and of course you'll want to do that with a regex that matches any tag and replaces it with nothing (that is, deletes it). Once you've done that, these lines look like title lines, except that they have space characters before them, and you can use a regex that matches one or more space characters to help match them. You can then capture each line (throwing away the leading white space by excluding it from the capture) and wrap it in `<title>` tags. You'll want to get rid of the paragraph tags that are wrapping the whole table of contents, since it isn't a paragraph, and replace it with something like `<toc>` (for "table of contents").

## Checking your results

Although you've added XML markup to the document, `<oxyen/>` remembers that you opened it as plain text, which means that you can't check it for well-formedness. To fix that, save it as XML with File → Save as and give it the extension ".xml". Even that doesn't tell `<oxyen/>` that you've changed the file type, though; you have to close the file and reopen it. When you do that, `<oxyen/>` now knows that it's XML, so you can verify that it's well formed in the usual way: Control+Shift+W on Windows, Command+Shift+W on Mac, or click on the arrow next to the red check mark in the icon bar at the top and choose "Check well-formedness".

## What to submit

We don't need to see the XML that you produce as the output of your transformation because we're going to recreate it ourselves anyway, but you do need to upload a step-by-step description of what you did. Your write-up can be brief and concise, but it should provide enough information to enable us to duplicate the procedure you followed.

If you don't get all the way to a solution, just upload the description of what you did, what the output looked like, and why you were not able to proceed any further. As always, you're encouraged to post any questions on the discussion boards, in this case in the regex forum.