Portfolio 2 Task: Wiki Markup (Easy)

We have seen that mark-up languages such as HTML allow us to annotate text to define how it should be displayed or printed. However, the mark-up tags used in HTML are intrusive and intimidating for people who are not IT-literate. In this easy task you will use *regular expressions* to develop a function called `wiki_markup` that allows text documents to be marked-up in a much simpler way. As its name suggests, our mark-up notation is based on the one that Wikipedia uses.

A simple mark-up notation

Wikipedia allows its contributors to create articles using a "lightweight" mark-up notation. Here we use a similar notation, but with a few modifications that make the markup-to-HTML translation process a little easier.

The equivalences between our Wikipedia-like mark-up notation (the supplied input) and HTML (the desired output) are as follows.

Description	**Wiki Markup**	**HTML tag**
An en-dash	`--`	**`&ndash;`**
An em-dash	`---`	**`&mdash;`**
Italic text	`''`*text*`''`	**`<i>`***text***`</i>`**
Boldface text	`'''`*text*`'''`	**`<b>`***text***`</b>`**
Major heading	==*text*==	**`<h1>`***text***`</h1>`**
Minor heading	===*text*===	**`<h2>`***text***`</h2>`**
Ordered list item	# *text* #	**`<li>`***text***`</li>`**
Unordered list item	∗ *text* ∗	**`<li>`***text***`</li>`**
"Normal" paragraph	[*text*]	**`<p>`***text***`</p>`**
Ordered list paragraph	[*list items*]	**`<ol>`***list items***`</ol>`**
Unordered list paragraph	[*list items*]	**`<ul>`***list items***`</ul>`**

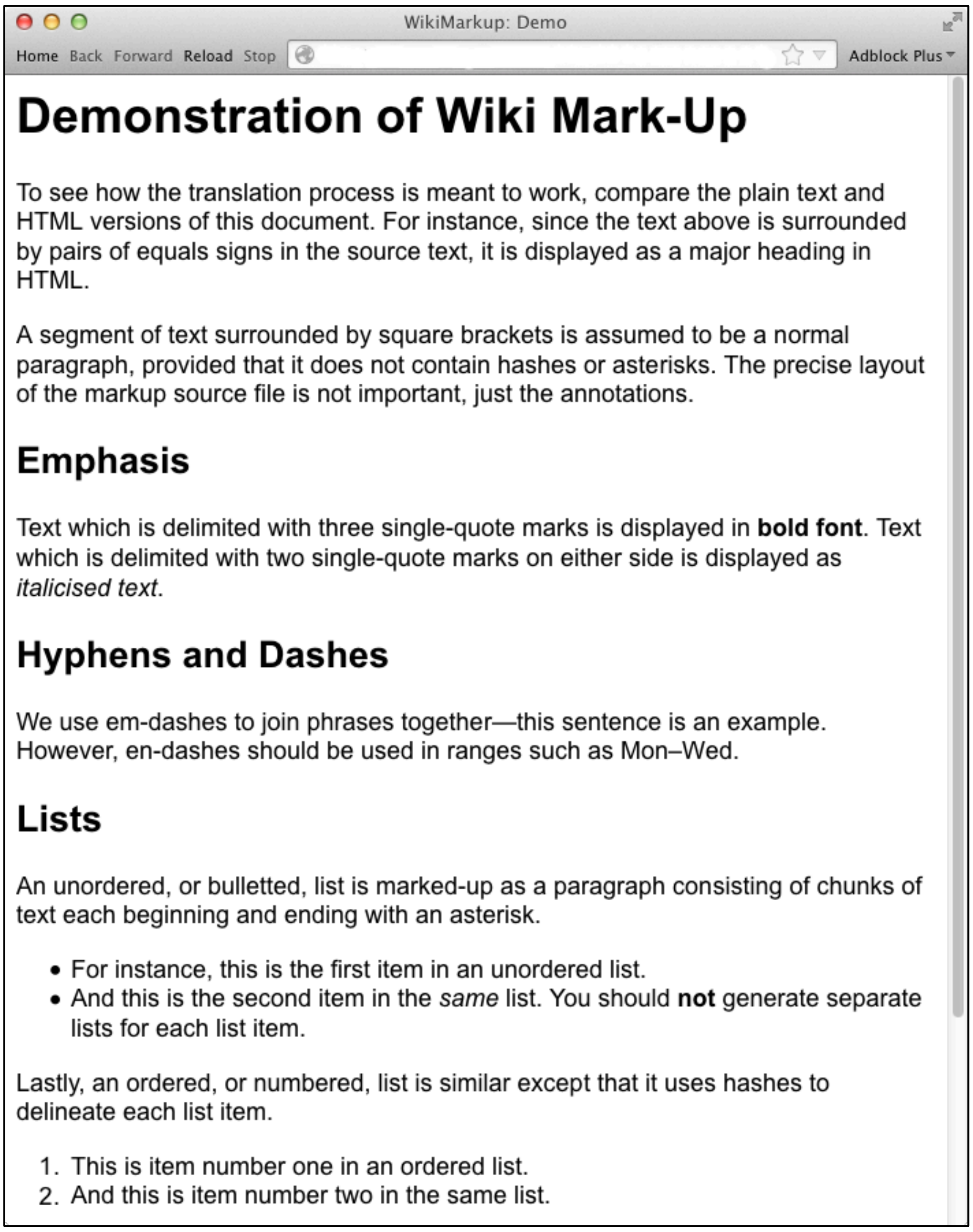
Note that spaces and newline breaks are not significant to our mark-up notation, so can be ignored or treated like ordinary characters. Also note that some of the text annotations in both the mark-up and HTML columns above are the same, so you need to take the context of these annotations into account when processing them.

Your job is to write a Python function, `wiki_markup`, which reads text files containing mark-up annotations, as per the middle column above, and produces corresponding HTML files, containing tags as per the column on the right. To do this you will need to use Python's regular expression module to make the necessary substitutions in the text.

For example, accompanying these instructions is a demonstration example, `Demo.txt` and `Demo.html`, to illustrate the necessary correspondence between the markup notation and the HTML code. The plain text mark-up file is as follows:

```
== Demonstration of Wiki Mark-Up ==

[ To see how the translation process is meant to work, compare the plain
text and HTML versions of this document.  For instance, since the text
above is surrounded by pairs of equals signs in the source text, it is
displayed as a major heading in HTML. ]

[ A segment of text surrounded by square brackets is assumed to be a
normal paragraph, provided that it does not contain hashes or
asterisks.  The precise layout of the markup source file is not
important, just the annotations. ]

=== Emphasis ===

[ Text which is delimited with three single-quote marks is displayed
in '''bold font'''.  Text which is delimited with two single-quote
marks on either side is displayed as ''italicised text''. ]

=== Hyphens and Dashes ===

[ We use em-dashes to join phrases together---this sentence is an example.
However, en-dashes should be used in ranges such as Mon--Wed. ]

=== Lists ===

[ An unordered, or bulletted, list is marked-up as a paragraph consisting
of chunks of text each beginning and ending with an asterisk. ]

[
  * For instance, this is the first item in an unordered list. *

  * And this is the second item in the ''same'' list.  You should
    '''not''' generate separate lists for each list item. *
]

[ Lastly, an ordered, or numbered, list is similar except that it uses
hashes to delineate each list item. ]

[
  # This is item number one in an ordered list. #

  # And this is item number two in the same list. #
]
```

Notice how the text is marked-up using our annotations to indicate headings, paragraphs, lists and emphasised text. When this text is processed using your `wiki_markup` function, these mark-up annotations should be replaced by corresponding HTML tags, as per the table on the first page, so that when the resulting HTML file is opened in a web browser it displays as a properly formatted document as follows.



Demonstration of Wiki Mark-Up

To see how the translation process is meant to work, compare the plain text and HTML versions of this document. For instance, since the text above is surrounded by pairs of equals signs in the source text, it is displayed as a major heading in HTML.

A segment of text surrounded by square brackets is assumed to be a normal paragraph, provided that it does not contain hashes or asterisks. The precise layout of the markup source file is not important, just the annotations.

Emphasis

Text which is delimited with three single-quote marks is displayed in **bold font**. Text which is delimited with two single-quote marks on either side is displayed as *italicised text*.

Hyphens and Dashes

We use em-dashes to join phrases together—this sentence is an example. However, en-dashes should be used in ranges such as Mon–Wed.

Lists

An unordered, or bulletted, list is marked-up as a paragraph consisting of chunks of text each beginning and ending with an asterisk.

- For instance, this is the first item in an unordered list.
- And this is the second item in the *same* list. You should **not** generate separate lists for each list item.

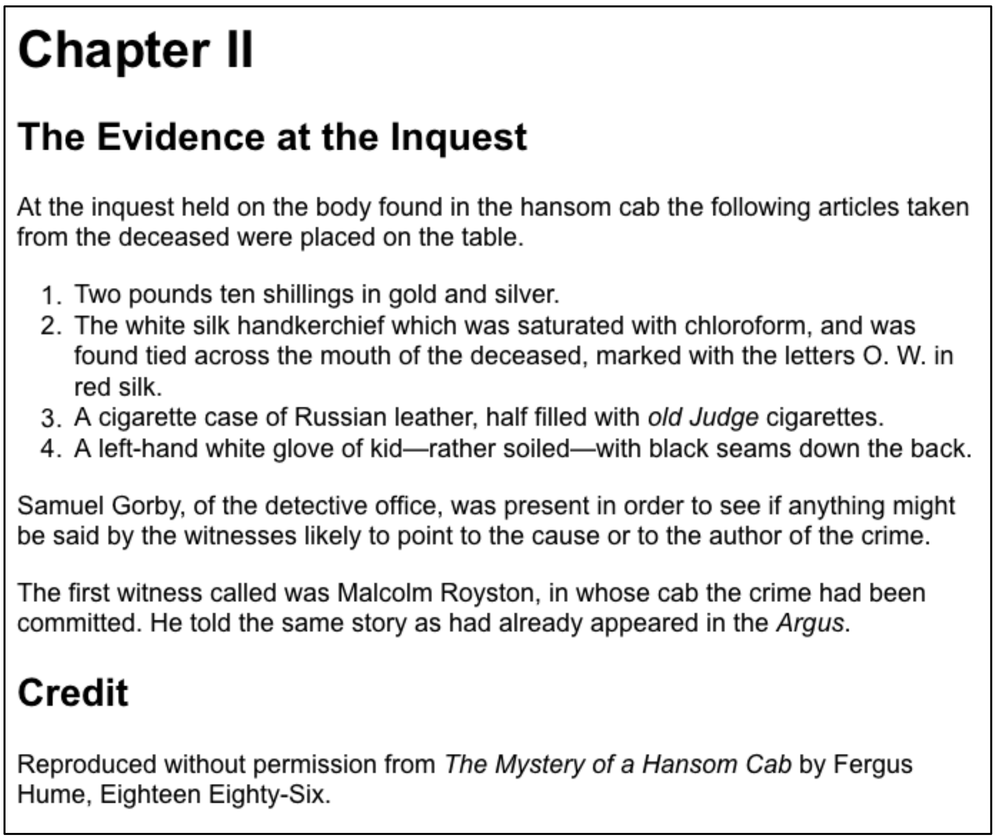
Lastly, an ordered, or numbered, list is similar except that it uses hashes to delineate each list item.

1. This is item number one in an ordered list.
2. And this is item number two in the same list.

Notice the correspondence between features of the annotated plain text and the displayed HTML document. We have provided a copy of the HTML file, so you should study it to see which HTML tags have been introduced to produce the document shown above. Your HTML document should have a title which includes the filename, ‘`Demo`’ in this case, and a separate ‘head’ and ‘body’ section.

NB: The `Demo.txt` file has been marked-up very consistently. However, in the various other test files we have provided, the layout of the annotated text has been deliberately made inconsistent because 'whitespace' in the source file is unimportant. You need to translate the mark-up annotations into HTML tags irrespective of the text file's layout.

As another example, here is how the HTML version of given Test #6 should appear when displayed in a web browser:



Chapter II

The Evidence at the Inquest

At the inquest held on the body found in the hansom cab the following articles taken from the deceased were placed on the table.

1. Two pounds ten shillings in gold and silver.
2. The white silk handkerchief which was saturated with chloroform, and was found tied across the mouth of the deceased, marked with the letters O. W. in red silk.
3. A cigarette case of Russian leather, half filled with *old Judge* cigarettes.
4. A left-hand white glove of kid—rather soiled—with black seams down the back.

Samuel Gorby, of the detective office, was present in order to see if anything might be said by the witnesses likely to point to the cause or to the author of the crime.

The first witness called was Malcolm Royston, in whose cab the crime had been committed. He told the same story as had already appeared in the *Argus*.

Credit

Reproduced without permission from *The Mystery of a Hansom Cab* by Fergus Hume, Eighteen Eighty-Six.

Specific tasks

You are required to implement a function called `wiki_markup` which accepts a single parameter, the name of a text file (without the '`.txt`' extension). When called with a filename *X* the function must do the following steps.

1. It must open and read the contents of text file *X*`.txt`. This can be done easily using Python's `open` and `read` functions, making sure you open the file in 'universal mode' as explained below.
2. It must transform the plain-text mark-ups to corresponding HTML tags as explained above. The easiest way to do this is to use the `sub` function in Python's regular expression module. The necessary transformations should be done one at a time through calls to `sub`. NB: The order in which you make the changes has a significant effect on the process because, for instance, square brackets appear in our mark-up notation for different kinds of paragraph. Therefore, you should *plan* your substitution strategy carefully before you start writing code.

3. It must write the final marked-up text to a new HTML file *X*.`html`. This is done easily by opening the new file in 'write mode' and calling Python's `write` function. You must include appropriate tags for an HTML document, i.e., `html`, `head` and `body`, and must include a `title` for the document as a whole.
4. Since there is a possibility that there is no file with the given name, or the required output file cannot be opened successfully, your `wiki_markup` function must return a Boolean result indicating whether or not the input file was read and the output file was written successfully. See the unit tests in the supplied template file for examples.

We have provided a template file `wiki_markup_Q.py` and several test files containing marked-up text. Each test file adds a new mark-up feature until the final one, which uses all of our mark-up annotations, so you should work through them in order. (The test files themselves contain some further hints, so it's worthwhile reading them!)

Note about newline markers

The text files for this task use DOS newline markers, since this is the format expected by Microsoft's Windows operating system. Linux and Apple Mac operating systems use different newline markers for text files, which can create problems with unexpected 'carriage return' characters appearing in the string (denoted '`\r`' in Python). Fortunately Python allows us to open files in 'universal' mode, in which case the interpreter automatically recognises the newline markers found in the file, regardless of the file format used. Therefore, when you open a text file for reading in this task, specify universal reading mode as follows:

`open(`*filename*`, 'U')`

You should do this regardless of which operating system you are using, Microsoft Windows, Apple Mac OS X or Linux.

Furthermore, we have defined the mark-up notation so that line breaks are not important, and can be effectively ignored or just treated like any other character. Recall that you can refer to a newline explicitly in a Python string as '`\n`'.

Development hints

- Since whitespace (blank spaces and newlines) is not significant to our markup notation, you can process the entire input file as one long string. Python's '`read`' function will return the entire contents of a text file as a single string, with the newline breaks appearing as the special character '`\n`'.
- You will want to use backslashes '\' in regular expressions to "escape" certain characters or to indicate a regular expression backreference. However, since Python also uses backslashes for special characters in strings we need to tell it not to interpret these characters as special when they appear in regular expression strings. The easiest way to do this is to write your regular expression strings as "raw" Python strings, i.e., "`r'...'`", so that escape characters in regular expressions that are also meaningful in Python are ignored by the interpreter.

Deliverables

The deliverables for this task are as follows:

- A Python file called `wiki_markup.py`, based on the provided `wiki_ markup_Q.py` template (with the '`_Q`' suffix removed). It must contain a function called `wiki_markup` which transforms plain text files to HTML files as described above.
- A completed "statement" at the beginning of the Python file to confirm that this is your own work and to identify your programming pair. (We can't award you marks if we don't know who you are!) Also you must indicate the percentage contribution made to the whole portfolio by both students. We do *not* expect a precise 50/50 split; an accurate and honest assessment is appreciated. This percentage will not affect your marks *except* in cases of extreme imbalances or disagreements.

As usual, your program code should be well laid-out and made easy to understand through appropriate commenting and choices of identifiers.