

## Software Engineering 265 Software Development Methods Fall 2019

Assignment 4 Due: Wednesday, December 4, 9:00 pm by submission via git (no late submissions accepted)

### Assignment Overview

The HTML language, which is used to describe the layout and content of web pages, has a famously verbose syntax: relatively simple formatting and layout instructions can often require several layers of bulky HTML tags. Modern HTML is very flexible for specifying visual aspects of the displayed content. However, extracting information from the HTML representation can be difficult. The goal of this assignment is to write a Python 3 `table_to_csv.py` program which converts HTML tables to a CSV representation.

The following sections describe HTML Tables, a Specification for the program you are expected to write as well as an example input HTML and expected CSV representation, along with some Implementation advice. We also provide a Testing component and a set of testing files. We close out with a section describing What you should Submit and the Evaluation scheme is also provided. Your code is expected to run **without warnings in the course lab** using Python version 3.

### HTML Tables

Tables in HTML are specified with the `<table>` tag. A brief tutorial on the `<table>` tag, along with interactive examples, can be found at [http://www.w3schools.com/html/html\\_tables.asp](http://www.w3schools.com/html/html_tables.asp). Note that whitespace in HTML is generally ignored: spaces, newlines and tabs are collapsed into a single space when the page is rendered. Line breaks are specified by the `<br />` tag. Consider the HTML table below (which appears as the first example in the tutorial linked above).

```
<table style="width:100%">
  <tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
  </tr>
  <tr>
    <td>Alfreds Futterkiste</td>
    <td>Maria Anders</td>
    <td>Germany</td>
```

```

    </tr>
    <tr>
        <td>Centro comercial Moctezuma</td>
        <td>Francisco Chang</td>
        <td>Mexico</td>
    </tr>
</table>

```

The table described by the HTML code above would be rendered by a web browser in a similar format to the table below.

<b>Company</b>	<b>Contact</b>	<b>Country</b>
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico

The `<tr>` and `</tr>` tags enclose the data for each row of the table, and the `<td>` and `</td>` tags enclose the contents of each cell. The `<th>` and `</th>` tags enclose the contents of header cells, but their use is optional (some authors use regular `<td>` tags for header cells). Cells of a table may contain HTML code, including other HTML tables.

Tag names are not case sensitive, so `<td>` and `<TD>` are both valid forms of the `<td>` tag. Any HTML tag may contain attributes that change its appearance.

The most common attribute in modern HTML is `style`. For example, make the contents of a particular cell boldfaced, the attribute `style="font-weight: bold;"` can be added to the `<td>` tag:

```
<td style="font-weight: bold;">Alfreds Futterkiste</td>
```

Additionally, any HTML tag may contain whitespace after the tag name, between attributes or before the closing `>` character. Whitespace is not permitted between the opening `<` character and the tag name. For example, `<td >` and `</td >` are valid tags, but `< /td>` and `< td >` are not.

Since whitespace in HTML is generally ignored, there is no requirement that the table be laid out in a readable way in the HTML code. The table in the example above could also be represented by the code below.

```
<table style="width:100%"> <tr> <th>Company</th> <th>Contact</th> <th>Country</th></tr>
<tr><td>Alfreds Futterkiste</td>
    <td>Maria Anders</td><td>Germany</td>
</tr><tr><td>Centro comercial Moctezuma</td><td>Francisco Chang</td> <td>Mexico</td></tr></table>
```

This assignment assumes the following extra constraints to any basic HTML table specification used as test input file.

- To be considered valid, a test input must be valid HTML. For example, tags like `<td>` can only occur inside of a `<tr>` tag, which in turn must be inside a `<table>` tag. All opening tags must have a matching closing tag (note that some HTML tags, like `<br />`, are singular and do not need a closing tag), and vice versa.
- Between the opening angle bracket (`<`) and closing angle bracket (`>`) of a tag, no other instances of closing angle brackets are permitted (including inside of attributes).
- Commas may not appear inside the data for a cell. However, other aspects of the HTML which are not cell content (such as the style attributes of `<td>` tags) may contain commas.
- Cells may contain any data, including other HTML tags, but may not contain nested `<table>` tags. The prohibition on comma use applies to all contents of each cell, including HTML tags. **In other words, if the comma character appears between the opening `<td>` tag and its matching `</td>` tag, the input will be considered invalid.**
- There is no requirement that each row of the table contain the same number of columns.
- Every HTML table must have at least one row.
- Every row of an HTML table must contain at least one cell.
- The `rowspan` and `colspan`, which are used to make cells span multiple rows or columns, are not permitted.

## HTML-to-CSV Converter

Your task is to write a Python 3 program called `table_to_csv.py` which reads HTML from standard input and outputs a CSV representation of each table in the input, including any header cells specified with `<th>` (if present).

The resulting CSV data will be printed to standard output in the following format:

TABLE 1:

<CSV data for first table in the input>

TABLE 2:

<CSV data for the second table in the input>

TABLE 3:

<CSV data for the third table in the input>

...

Your implementation may assume that the input table complies with the constraints given in the previous section, and must also meet the following requirements.

- All runs of one or more spaces, newlines, tabs, or other whitespace should be collapsed into a single space.
- Within a table cell, all HTML tags are to be left intact.
- The contents of each table cell should be stripped of all leading and trailing whitespace before being output. For example, the cell ` Lemon Meringue </td>` should be output as `Lemon Meringue` (note that the multiple spaces between the two words are also collapsed into one space). |
- Every row of the output CSV spreadsheet must contain the same number of columns (recall that the number of columns in a row of a CSV spreadsheet is the number of commas in the row minus one). If the rows of the HTML table contain a differing number of columns, then the number of columns in the output spreadsheet should be equal to the number of columns in the row of the input table with the largest number of columns. Other rows should be padded with blank cells to meet the column requirement.

As an example, consider the following input HTML table below:

```
<table><tr>
<th>Student Number</th><th>Student Name</th><th>Major</th>
<th>A1 mark</th><th>A2 mark</th></tr >
<tr><td>V00000001</td><td></td><td></td><td>10</td>
<td>11</td>
</tr>
<tr><td>V00123456</td><td>Alastair Avocado</td>
<td>Psychology</td><td>12</td><td></td></tr>
<tr >
<td>V00123457</td >
<td>Rebecca Raspberry
```

```

</td><td>Computer Science</td><td>17</td><td>14</td></tr>
<tr><td>V00314159</td><td>Fiona Framboise</td>
<td style="font-family: monospace; font-size: 20pt; font-weight: bold;">
Computer Science

</td>
<td> </td><td>17</td></tr>
<tr><td>V00654321</td><td>Meredith Malina</td>
<td style="color: red;">Software Engineering</td><td>18</td><td>12</td></tr>
<tr><td>V00654322</td><td>Hannah Hindbaer</td><td>Physics</td><td>15</td><td>18</td></tr>
<tr><td>V00951413</td><td>Neal Naranja</td><td>Anthropology</td><td>15</td><td>15</td></tr>
</table>

```

When provided as input to a correct HTML-to-CSV implementation, the HTML table above would be converted to the following CSV representation.

TABLE 1:

```

Student Number,Student Name,Major,A1 mark,A2 mark
V00000001,,,10,11
V00123456,Alastair Avocado,Psychology,12,
V00123457,Rebecca Raspberry,Computer Science,17,14
V00314159,Fiona Framboise,Computer Science,,17
V00654321,Meredith Malina,Software Engineering,18,12
V00654322,Hannah Hindbaer,Physics,15,18
V00951413,Neal Naranja,Anthropology,15,15

```

## Implementation Advice

Since HTML allows such a wide variation in the structure and formatting of tags, the use of regular expressions to match each tag pair is encouraged. However, you are not required to use regular expressions (or any other particular implementation technique, as long as your code is valid Python 3). If you use regular expressions, be aware of the following points.

- By default, the ``.'` specifier does not match the newline character (``\n'`), so if you are searching for something which crosses a line boundary, it will not match. For example, the pattern ``A.*B'` would match ``Axy z B'` but not ``Axy\n z B'` by default. Since whitespace in HTML can be collapsed to a single space, you can remedy this problem by replacing all newlines characters with spaces. You can also use the ``re.DOTALL'` flag when performing regular expression matching, which will cause newlines to be matched by the ``.'` specifier. Consider the interactive Python 3 session below, which contains examples of both methods.

```
>>> s1 = 'Axy z B'
>>> s2 = 'Axy\n z B'
>>> re.match('A.*B',s1)
<_sre.SRE_Match object; span=(0, 8), match='Axy z B'>
>>> re.match('A.*B',s2)
None
>>> re.findall('A.*B',s1)
['Axy z B']
>>> re.findall('A(.*?)B',s1)
['xy z ']
>>> re.findall('A(.*?)B',s2)
[]
>>> re.findall('A(.*?)B',s2, re.DOTALL)
['xy\n z ']
>>> s3 = s2.replace('\n', ' ')
>>> s3
'Axy z B'
>>> re.findall('A(.*?)B',s3)
['xy z ']
```

- Since HTML tag names are not case sensitive, you may want to use the `re.IGNORECASE` flag to enable case-insensitive matching. Consider the interactive session below.

```
>>> x = 'abc'
>>> y = 'Abc'
>>> z = 'A-----C'
>>> re.findall('a.*c',x)
['abc']
>>> re.findall('a.*c',y)
[]
>>> re.findall('a.*c',z)
[]
>>> re.findall('a.*c',y,re.IGNORECASE)
['Abc']
>>> re.findall('a.*c',z,re.IGNORECASE)
['A-----C']
```

Note that if you want to use multiple flags (such as both `re.DOTALL` and `re.IGNORECASE`), you can combine them with the bitwise-OR operator (for example,

```
re.findall('a.*c',z, re.IGNORECASE|re.DOTALL)').
```

## Constraints

You may only use python3 modules that are installed on the lab computers. If a python3 module is not installed, you may not use that module in your code.

## Test Inputs

You should ensure that your program appropriately handles error cases (such as files which do not exist - you “know the drill” from all

the previous assignments) and does not produce errors on valid inputs. Since thorough testing is an integral part of the software engineering process, your `git` repo also provides a test folder which contains 3 sub-folders each containing an HTML test input file (containing various HTML tables) used to evaluate assignment 4, together with the expected output csv representation (`output.txt`). The test input files must be read from `stdin`, as in the following example usage pattern:

```
python3 table_to_csv.py < input.html > your_output.txt
```

The test suite and correctness checker will be run exactly as in previous assignments -- the output you generate should strive to match the expected output (i.e. `diff your_out1.txt out1.txt`), and you can use the same techniques you've established for ensuring quality and correctness.

## What you must submit

- Python source-code named `table_to_csv.py` which contains your solution for assignment #4
- Ensure your work is committed to your local repository in the provided assignment4 folder and pushed to the remote before the due date/time. (You may keep extra files used during development within the repository.) The assignment will be marked based on the last version of your program pushed before the deadline.

## Evaluation

The teaching staff will primarily mark solutions based on the input files provided for this assignment, though additional files might also be used. Students must adhere to the command execution and output formatting outlined in this assignment. There will be no demos for this assignment.

In addition to automated testing, your code will be evaluated based on:

- Proper error handling
- Good coding practices (i.e. good variable/function naming, use of functions when appropriate, limited globals, etc.)
- All the usual items we evaluate in this course!

## Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Murray). If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted programs.)