CIT 590 Homework 5 – HTML Resumes

Purposes of this assignment

- Reading from and writing to files
- Scraping information from a text file
- Basic HTML usage

General problem specification

A website is made up of an HTML page. This HTML page is simply a text file with a certain format. Taking advantage of this fact, one can take an HTML template and create multiple pages with different values stored. This is exactly what we are going to do in this HW. Many websites use these kinds of scripts to mass generate HTML pages from databases, but we will just think of our text file as our database. Our goal will be to take a resume that is a simple text file and convert it into an HTML file that can be displayed by any web browser.

Is knowledge of HTML necessary to complete this assignment?

You do not need to know much HTML to do this assignment. In class, I will cover the basics that might be helpful. The TAs will also go over some examples during the recitations, so going to the recitation can give you a little bit more of an appreciation for what this format is.

You can also do the assignment by just understanding that HTML is the language that your browser understands. It is a tag-based language where each tag provides some basic information for how the browser renders the text that you write between them.

For example, **`<h1>Rado</h1>`** will be rendered by your browser as a heading (saying “Rado”) with large font. **`<h1>`** is the beginning of the heading. And **`</h1>`** indicates the ending of a heading.

An HTML webpage is typically divided into a head section and a body section. We have provided you with a basic website template. We want you to retain the head section and write only the body section via your Python program.

For more details about HTML your best resources will be to show up for class/recitation or to use the **`w3schools`** websites which can be found at **`www.w3schools.com`**. That website has a ton of information. In any case, really all that you will need for this assignment is the tag documentation.

Also, note that the first part of the assignment where you just have to read from a file has absolutely nothing to do with HTML.

The input text file

We will read a simple text file that is supposed to represent a student’s resume. The resume has some key points but can be somewhat unstructured. In particular, the order of some of

the information can definitely be different for different people. But these are the things that you definitely DO know about the resume:

- Every resume will have a name which will be written at the top. The top line in the text file has to be just the name.
- There will be a line in the file which contains an email address. It will be a single line with just the email address and nothing else.
- Every resume will have a list of projects. Projects are listed line by line below a heading called Projects.

  So an example of what you will see in your resume files is

  ```
  Projects
  Worked on the big huge robot simulator
  Built a bridge across the Pacific Ocean
  Designed an efficient algorithm to sort arrays faster than Superman
  ```

  The list of projects ends with a single line that looks like '`----------`'. That is, it will have at least 10 minus signs. While this is a weird requirement; we are imposing it to actually make the assignment easier for you. So please go with it.

- Every resume will have a list of courses done. Courses are listed in the following format:

  `Courses - CIT590, AB120` or `Courses :- Ray Tracing, Making money by lying`

  You are allowed to assume that every single resume will just have this comma-separated list of courses with the word Courses being there in front of them. You do want to allow for some kind of punctuation marks after the word Courses. So your program should be able to look at the above example and then extract the courses without including the '-' sign or the ':-' or any such punctuation that is in between the word Courses and the actual data we care about.

  Every course begins with a letter of the English alphabet regardless of whether we are putting the course ID or the course name in our resume.

- Every resume will have one or more degrees. A degree is to be found in a single line in the text file that contains the word 'University' and also contains one of the words 'Bachelor', 'Master' or 'Doctor'. Example:

  `Bachelor of Science in Business Administration - University of Python` or `Master of Technology in Chemical Engineering - University of Mumbai`

  Be careful with this. In particular, a project done in a university should not be incorrectly considered a degree.

Your first step in this assignment is to create this type of file. We provide an example in the HW folder.

Please do not just blindly copy our resume. Write your own. It does not have to be totally accurate but the HW is going to be more fun if you make it close to reality.

Functions for reading

You have to write one function for each piece of information that you want to extract from this text file. Contrary to previous assignments, in this assignment we will not tell you what to call the functions or what arguments to pass to them.

What we will tell you is that you will be graded on what you call the functions, how modular your code is and most importantly whether you have unit tested your functions or not.

Since the resume file is pretty small, these functions are best done by reading the file into memory and then parsing the file using list and string manipulations. Keep in mind that functions that perform input/output are difficult to test, so limit your file reading to a single function so that you can better unit test the rest of your functions.

Detecting name

This one is easy. Just extract the first line. The one extra thing we want you to do, just for practice, is to raise an error if the first character is not '**A**' through '**Z**'. So yes, in this case, your program will crash. Just provide a message saying that the first line has to be just the name with proper capitalization.

Detecting emails

For detecting emails here is what you need to do.

1. Look for a line that has the '**@**' character.

2. Also make sure that the last four characters of the email are either '**.com**' or '**.edu**'. We want those characters in those specific cases. So '**.COM**' and '**.EdU**' and all variants like that will not be allowed.

3. Make sure there is a string that begins with a normal lowercase English character between the '**@**' and the ending ('**.com**' or the '**.edu**').

These rules will work for **rivanov@seas.upenn.edu** but will not work for **@rivanov** or **rivanov@python.org**. We are fully aware that these rules are inadequate. However, we want you to use these rules and only these rules.

PLEASE DO NOT GOOGLE FOR A FUNCTION FOR THIS. Googling for solutions to your homework is an act of academic dishonesty and in this particular case you will get solutions involving crazy regular expressions, which is a topic we have not talked about in class. In general, your code should never involve a topic that we have not touched in class at all.

Remember if you already have expert knowledge of Python, you should not be in this class anyway.

Detecting courses

Look for the word Courses in the file and then extract the line that contains that word. Then make sure you extract the correct courses. In particular any random punctuation after the word Courses and before the first actual course needs to be ignored. You are allowed to assume that every course begins with a letter of the English alphabet.

Detecting projects

Look for the word Projects in the file. Each subsequent line is a project, until you hit a line that looks like this '`----------`'. That is NOT an underscore. It is (at least) ten minus signs put together. Declare that you have reached the end of the projects section if and only if you see a line that has at least 10 minus signs one after another. If you detect a blank line between project descriptions, ignore that line.

Detecting education

Look for the word University or university and the words that are indicative of a degree. Grab any line that has those words. Please grab the entire line.

Writing HTML

Once you have gathered all the pieces of information, we want you to write it in HTML format. Here are the steps for that. Again, please put each one of these steps into a separate function so that you can debug and test it more efficiently.

Save the file **`resume.html`** that is provided to you in the same directory as your code. Open that file in an editor that understands HTML. Note that there is empty **`<body>`**. We are going to go in and fill the body. In order to do this, you will need to write a function which opens the **`resume.html`** file that we have provided and then writes information to it bit by bit. Also since we want to write the body part of this HTML file, we want to leave the head section untouched. In order to do that, you will probably need the following code. Note that you will have to change the path so that it points to wherever you put your **`resume.html`** file.

```
f = open('C:\Users\Rado\Desktop\resume.html', 'r+')
lines = f.readlines()
f.seek(0)
f.truncate()
del lines[-1]
del lines[-1]
f.writelines(lines)
```

There are a few lines over here that are worth explaining since we haven't covered all of this. **`f.seek(0)`** moves the file pointer back to the top of the file. **`f.truncate()`** deletes everything

from the file from that point onwards. The combination of those 2 lines effectively delete every single line in the file. Why are we doing this? Our file looks something like

```
<html>
<head>
<title>Resume</title>
</head>
<body>
</body>
</html>
```

What we want to do is stuff our resume content within the body tags to make it look like this:

```
<html>
<head>
<title>Resume</title>
</head>
<body>
all the resume content goes over here
</body>
</html>
```

In order to write proper HTML we will need the following helper function: **`surround_block(tag, text)`**. This is a function that surrounds some text in an HTML block. For example, something that writes **`<div> </div>`** with some text inside it. Now break the writing of the resume into the following steps:

1. In order to format the resume a little bit we have to make sure that all the text is enclosed within the following lines:

   ```
   <div id="page-wrap">
   </div>
   ```

   Since we can only write things line by line we will write the first line now, then fill up the actual content in a few steps and write the final 3 lines (look at item 6 below) to close the HTML file properly. So this initial step just writes one line which is **`<div id="page-wrap">`**.

2. The basic information section looks like

   ```
   <div>
   <h1>
   Rado
   </h1>
   <p>
   Email: rivanov@seas.upenn.edu
   </p>
   ```

```
</div>
```

Think about how you can do that. In particular, think about how **surround_block** can be used to achieve this. Write a function to make this intro section of the resume and then write it out to a file (make sure your file writing is done in a separate function because it is hard to unit test).

3. For the education section you will have to produce output similar to the following. Assume that you have got the data about the degrees by reading through the original file and stored that in some data structure (decide whether you want to use list, set, tuple or dictionary). Note that the **<ul>** tags contain an unordered list, where the **<li>** tags contain a list item.

```
<div>
<h2>Education</h2>
<ul>
<li>
Masters in Java
</li>
<li>
Bachelors in Typing
</li>
</ul>
</div>
```

4. The projects section is actually quite similar to the education section. The only difference is that since project descriptions can be more verbose we want to wrap everything in a paragraph tag **<p>**. For an example, here is the kind of html that we want you to generate for the projects section. Write a function for doing the projects section. The arguments for this section should take in a list of projects.

```
<div>
<h2>
Projects
</h2>
<ul>
<li>
<p>
Worked on the foreground background segmentation problem, which
is a classic problem in computer vision.
</p>
</li>
<li>
<p>
Compiled data on grade inflation. Published several papers on
this topic in the journal of ivy league humor.
</p>
```

```
</li>
</ul>
</div>
```

Do not worry about the indentation.

5. For the courses section, we just want to list the courses. We do not want to create bullet points. We also want the heading to be a bit smaller in size. So here is an example of generated html.

   Write a function that takes in courses (you can decide whether you want a list of courses or a string of courses) and then writes out the html of the form below into the file.

```
<div>
<h3>
Courses
</h3>
<span>
Algorithms, Forex training
</span>
</div>
```

6. After writing all this content we just need to remember to close the HTML tags. To do this, we need to write the following 3 lines to the file:

```
</div>
</body>
</html>
```

Formatting your HTML file

If you are interested in converting your HTML file into more readable HTML, you can use the following website: **`http://www.dirtymarkup.com/`**. Just copy your HTML and hit clean and you will obtain cleanly formatted code. Note that this will not affect your grade in any way.

Suggestion for working in pairs

This is an assignment where there is a natural split for people to work in a pair. One person can scrape the text file for the information. The other person can write the functions that will write to an HTML file. If you work in pairs, please indicate both people in the header of your python file **as well as in the Canvas comment section**. One Canvas submission per pair is enough.

Evaluation

You will get evaluated on the following dimensions. Please note that we are happy to give you feedback about your design in office hours. It is important that you understand that in this assignment there is more to it than just getting your code to work.

1. Does your program successfully convert a resume text file into a resume HTML file?
2. Does that HTML file get rendered in a browser? Test it out in at least 2 different browsers (hopefully everyone has multiple browsers on their computer these days).
3. Did you sufficiently test your program?
4. How did you approach reading from and writing to files? Since we do not give you specific functions, we need to evaluate your design. The points over here will be a combination of your style, your modularity, the kind of functions you chose write, the documentation for each function, etc.

Submission and due date (not a Tuesday!)

Write all your Python code in a file called **`makeWebsite.py`**. Write your unit tests in a file called **`makeWebsite_tests.py`**. Finally, submit both the text file that you used to make your resume and the HTML file that got generated. Call them **`resume.txt`** and **`resume.html`**, respectively. We do not need a nicely formatted HTML file. We just need the file that your program produced. Also, if we need any other files to run your unit tests, please submit them as well.

Zip all your files and turn them in to Canvas before **11:59pm Thursday, October 19**.