CIT 590: Spring 2020
Homework 5

HW deadline as per Canvas.

This homework deals with the following topics:

- Reading and writing files
- Scraping and parsing information from a text file
- Very basic HTML

General Problem Specification

The basic skeleton of a website is an HTML page. This HTML page is 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. We will use a sample text file as our 'database'.

Our goal will be to take a resume in a simplistic text file and convert it into an HTML file that can be displayed by a web browser.

What is HTML?

You do not need to know much HTML to do this assignment. You can do the assignment by just understanding that HTML is the language that your browser interprets to display the page. It is a tag-based language where each set of tags provides some basic information for how the browser renders the text that you write between them.

For example, `<h1>The Beatles</h1>` will be rendered by your browser as a heading with large font. `<h1>` indicates the beginning of the heading and `</h1>` indicates the ending of the 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 resource will be to use the w3schools website which can be found here: www.w3schools.com. This website has a ton of information and provides all of the common HTML you'll need to know for this assignment.

Input Test 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 will definitely be different for different people.

The following 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 will contain 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". An example of what you might see in a resume file is something like this:

```
Projects
Worked on big data to turn it into bigger data
Applied deep learning to learn how to boil water
Styled web pages with blink tags.
Washed cars …
----------
```

- 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.
- Every resume will have a list of Courses. Courses are listed like:

```
Courses - CIT590, AB120
```

OR

```
Courses :- Pottery, 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 mark(s)* after the word "Courses", and before the list of 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're interested in.

The first step in this assignment is to create a sample resume that conforms to the format described above. We're providing an example in the HW folder on Canvas, but

please do not just blindly copy our resume. Write your own sample file in addition to the one provided. It does not have to be totally accurate but the HW is likely to be more fun if you make it close to reality.

Functions for Parsing the File

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

When we grade, we'll look at how modular your code is and how you decided to break up the functionality into separate functions. We'll also look at how you named the functions, what arguments they take, and how well you unit test the functions.

Here's a basic breakdown of the functionality required to read the file into memory, parse each section of the file to extract the relevant information, and write the final HTML-formatted information to a new file.

Reading the File

- Since the resume file is pretty small, write a function that reads the file and stores it in the program's memory.
- Then, you can use list and string manipulations to do all of the other necessary work.

Detecting the 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 a *RuntimeError* if the first character in the name string is not an uppercase letter (capital 'A' through 'Z').
- So yes, in this case, your program will crash. You must provide a message saying that the first line has to be a name with proper capitalization. We specifically want you to do this by raising a *RuntimeError*.
- For example:
  `Brandon Krakowsky` is a valid name
  `brandon Krakowsky` is not a valid name
- Another thing to note is that the name on the first line could have leading or trailing whitespace.

Detecting the Email

- Look for a line that has the '@' character.
- Also make sure that the last four characters of the email are either '.com' or '.edu'.
- Make sure the email string begins with a normal lowercase English character between the '@' and the ending ('.com' or the '.edu')
- There should be no digits or numbers in the email address.
- The email string could have leading or trailing whitespace.
- These rules will accommodate lbrandon@wharton.upenn.edu but will not accommodate @lbrandon or lbrandon@python.org or lbrandon2@wharton.upenn.edu
- For example:
  `lbrandon@wharton.upenn.edu` is a valid email
  `lbrandon@wharton2.upenn.com` is not a valid email
  `lbrandon2@wharton.upenn.com` is also not a valid email
- We are fully aware that these rules are inadequate. However, we want you to use these rules and only these rules.
- If an e-mail string is not found based on the given rules, consider the e-mail address to be missing. This means your function should return an empty string.
- 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 haven't yet discussed in class. (In general, your code should never involve a topic that we have not discussed in class.). Plus, you can easily achieve the required functionality without the use of a regular expression.

Detecting the 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.
- Note that the word "Courses", the random punctuation, or individual courses in the list could have leading or trailing whitespace.

Detecting the Projects

- Look for the word "Projects" in the file.
- Each subsequent line is a project, until you hit a line that looks like '----------'. This is NOT an underscore. It is (at least) ten minus signs put together. 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 the other.
- If you detect a blank line between project descriptions, ignore that line.
- Also note that certain projects could have leading or trailing whitespace.

Writing the HTML

Once you have gathered all the pieces of information from the text file, we actually want you to programmatically write HTML. Here are the steps for that:

- Start by saving the file *resume-template.html* that is provided in the same directory as your code.
- Preview the file in a text editor that does HTML syntxsax highlighting (e.g. Sublime Text). Notice that there is an empty `<body>`. You are going to programmatically copy the HTML in *resume-template.html*, fill in the empty `<body>` with the resume content, then write the final HTML to a new file *resume.html*.
- More specifically, your Python code will do the following:
    - Open and read *resume-template.html*
    - Read every line of HTML
    - Remove the last 2 lines of HTML (you'll programmatically add these back later)
    - Add all HTML-formatted resume content
    - Add the last 2 lines of HTML back in
    - Write the final HTML to a new file *resume.html*
- Why are we doing this? Because the HTML in *resume-template.html* looks something like this, and we need to start by removing the last two lines of HTML (closing `</body>` and `</html>` tags).

```
random header stuff
<html>
<head> lots of style rules we won't worry about
</head>
<body>
</body>
</html>
```

- We want to put our resume content in between the body tags to make it look

like this:

```
random header stuff
<html>
<head> lots of style rules we won't worry about
</head>
<body>
HTML-formatted resume content goes here
</body>
</html>
```

In order to write proper HTML you will need to write the following helper function:

def surround_block(tag, text):

- This function surrounds the given text with the given HTML tag and returns the string
- For example, *surround_block('h1', 'The Beatles')* would return

  `'<h1>The Beatles</h1>'`

You're going to display the email address in your webpage as an active email link. The proper way to create a link in HTML is to use the `<a>` and `</a>` tags. The `<a>` tells where the link should start and the `</a>` indicates where the link should end. Everything between these two tags will be displayed as a link and the target of the link is added to the `<a>` tag using the *href* attribute.

For example, a link to google.com would look like this:
Click here to go to Google

The `<a>` tag also provides the option to specify an email address as the target of the link. To do this, you use the "mailto: email address" format for the *href* attribute.

For example, an email link to abc@example.com would look like this:
Send Email

In order to write proper HTML for the email link, you will need to write the following helper function:

def create_email_link(email_address):

- This function creates an email link with the given email_address
- To cut down on spammers harvesting the email address from your webpage, this function should display the email address with an [aT] instead of an @
- For example, *create_email_link('tom@seas.upenn.edu')* would return

```
'<a href="mailto:tom@seas.upenn.edu">
tom[aT]seas.upenn.edu</a>'
```

- Note: If (for some reason) the email address does not contain @, use the email address as is and don't replace anything

Now break the writing of the resume into the following steps:

1. In order to format the resume content we have to make sure that all the text is enclosed within the following tags:

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

Since we typically write things line by line, this initial step will write the 1st line above. We'll then fill in the actual resume content in multiple steps, then write the 2nd line above, then write the final 2 lines of HTML to close the HTML file properly. So, this initial step just writes the 1st line above.

2. The basic information section needs to look like:

```
<div>
<h1>Brandon Krakowsky</h1>
<p>Email: <a href="mailto:lbrandon@wharton.upenn.edu">
lbrandon[aT]wharton.upenn.edu</a></p>
</div>
```

Think about how you can do that. In particular think about how the *create_email_link* and *surround_block* functions can be used to achieve this. Write another function to make this entire intro section of the resume and then write it out to a file.

3. For the projects section you will have to produce output like the following. Assume that you have the data about the projects by reading through the original file and stored that in some data structure (decide whether you want to use list, set, tuple or dictionary)

```
<div>
<h2>Projects</h2>
<ul>
<li>built a robot</li>
<li>fixed an ios app</li>
</ul>
</div>
```

4. For the courses section, we actually 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:

```
<div>
<h3>Courses</h3>
<span>Algorithms, Race car training</span>
</div>
```

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, in the form above, to the file.

5. After writing all of 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>
```

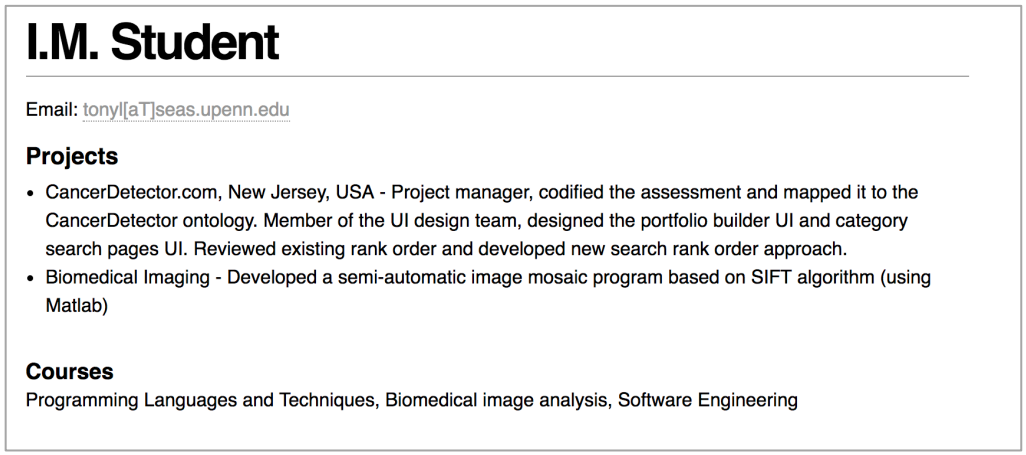
6. And most importantly, you need to close the file -- otherwise it will NOT save!

Unit Testing

We're providing you with a starter unit testing file *make_website_test.py*. Currently, it contains only two test functions, *test_surround_block* and *test_create_email_link*, since these are the only specific functions that we've asked you to write. Make sure you pass all of the tests in the unit testing file, plus, be sure to write other test functions to test the functions in your program.

Resume Files

Using the *resume.txt* file that we provided, the generated *resume.html* file should display something like this:



I.M. Student

Email: tonyl[aT]seas.upenn.edu

Projects

- CancerDetector.com, New Jersey, USA - Project manager, codified the assessment and mapped it to the CancerDetector ontology. Member of the UI design team, designed the portfolio builder UI and category search pages UI. Reviewed existing rank order and developed new search rank order approach.
- Biomedical Imaging - Developed a semi-automatic image mosaic program based on SIFT algorithm (using Matlab)

Courses

Programming Languages and Techniques, Biomedical image analysis, Software Engineering

We're also providing other versions of the *resume.txt* file that your program should be able to handle.

What to Submit

You will submit the following 4 files:

1. *make_website.py*: the program you wrote
2. *make_website_test.py* [and *other .txt files for testing*]: the unit tests you wrote for all your functions and if you created other .txt files (e.g. different versions of your resume) to be read by your main program or your unit tests, please include those in the .zip file. Part of grading your assignment will be to run your unit tests. We want to make sure they pass if they reference other .txt files.
3. *resume.txt*: the text file you created to be read by your program
4. *resume.html*: the resume file that was generated when you ran your program

Please zip all 4 of these files and name the .zip file using your PennKey. For example, Brandon's submission would be *lbrandon.zip*. Please, no .rar files!

Evaluation

It is important that you understand that in this assignment there is more to it than just getting your code to work. We are happy to give you feedback about your design during office hours.

1. Functionality – 15 points
    a. Does your program successfully convert a resume text file into a resume HTML file?
    b. Can that HTML file be rendered in a web browser? Test it out in at least 2 different browsers to confirm this.
2. Unit Testing – 10 points
    a. Did you write a test for each function you wrote? An exception to this rule would be a function that writes to a file. This is difficult to test with unit testing.
    b. Are you doing a good job of testing all the different cases for each function? You should be testing both typical examples and edge cases.
3. Design and Style – 15 points
    a. Since we do not give you specific functions, we need to evaluate your design.
    b. Did you follow style conventions as defined in this course? Do all functions have docstrings and non-trivial lines of code have comments?
    c. Did you use the best data structures and variable types for each situation?
    d. Did you simplify the logic of your functions? In other words, if it is possible to do the same thing with fewer lines of code, did you do so?