CSE/ISE 337 – Scripting Languages
Spring, 2023
Assignment 02 – Regular Expressions

Assigned: Tuesday, 02/20/2023
Due: **Tuesday, 03/06/2023, at 11:59 PM**
Total Problems: 6, Total Points: 70

Submission Instructions:

1. Submit all of your solutions in a single file named: `cseise337_a02.py`
2. Make sure you include your name, netid, and Student ID number as comments at the top of your submission file.
3. Please label each separate solution with the problem number in a comment.
4. Make sure you use the specified, file, function, and class names (and any other names specified in the instructions). Grading may be partially automated (by importing your code to run against our test cases) and the use of incorrect names will interfere.
5. This is an individual programming assignment. Any collaboration on coding will be considered a violation of academic honesty.
6. Every problem must be solved with the use of regular expressions, but you do not have to solve every problem *only* with regular expressions.

Problem 1 (10 points): `highlight(pattern, string)`

Write a Python function named `highlight()`, which takes a raw string specifying a regex pattern as the first argument, called *pattern*, and a string as the second argument, called *string*. The function will compile *pattern* and apply the compiled pattern to *string*. If a match is found, then the function will return a string in which the matched substring of *string* is enclosed in angle brackets. If no match is found, the function should return None.

For example, suppose that:

```
pattern = r'[a-zA-Z]\d'
string = "ShepardN7"

returns: "Shepard<N7>"
```

Test Cases:
```
pattern = r'\bbook\B'
strings = ["bookstore", "booking", "textbooks",
           "'returned books'", "audiobook"]
```

Problem 2 (10 points): `highlight_all(pattern, string)`

Write a Python function named `highlight_all()`. Like the `highlight()` function from Problem 1, this function will take a raw string, named *pattern*, and a target string, named *string*. This function will compile *pattern* and apply it to *string*. However, the `highlight_all()` function will return a string in which all substrings that match *pattern* will be enclosed in angle brackets. If there are no matches, then the function should return None.

For example, suppose that:

```
pattern = r'o\w+'
string = "I'm Commander Shepard and this is my favorite store on
the Citadel."

returns: "I'm C<ommander> Shepard and this is my fav<orite>
st<ore> <on> the Citadel."
```

Problem 3 (10 points): ruin_a_webpage(filename)
Write a Python function named ruin_a_webpage(). This function takes a single string argument named *filename*. *filename* will contain the name of an HTML file. Check the extension on the file using a regular expression. If the extension is not ".html" or ".htm", then the function should return None.

Otherwise, the function should read in the HTML file and do the following:

1. Use regular expressions to find content enclosed in paragraph tags, <p> and </p>, remove the paragraph tags, and then append 2 line break tags to the content, <br><br>
2. Remove all matched pairs of open and close span tags, <span> and </span>
3. Write the modified HTML content to a file called, "ruined.html"

For example, if the input file contains:

```
<html>
<body>
<p>trees of green, red roses too</p>
<p>I <span><span>se</span></span>e them bloom for me and you</p>
<p>(what <span><span>a</span></span> <span><span>wonderful</span>
</span>world)</p>
<p></p>
</body>
</html>
```

Then the output file, "ruined.html", should contain:

```
<html>
<body>
trees of green, red roses too<br><br>
I see them bloom for me and you<br><br>
(what a wonderful world)<br><br>
<br><br>
</body>
</html>
```

Problem 4 `(10 points): decompose_path(path)`
The shell environment variable $PATH contains a delimiter-separated sequence of directory paths (the delimiter varies between platforms). The shell uses the content of this variable to determine where to look for executables that correspond to commands entered at the command line. You can view this on Linux using the command: `echo $PATH` (using Windows CMD it's: `echo %PATH%`).

Write a function named `decompose_path()` that takes a single string argument, named *path*. *path* will contain the contents of the PATH environment variable as returned by the `echo $PATH` command. Use a regular expression to create a list of each directory in PATH. The function should return that list.

For example:

```
path =
"/usr/openwin/bin:/usr/ucb:/usr/bin:/bin:/etc:/usr/local/bin:/us
r/local/lib:/usr/shareware/bin:/usr/shareware/lib:."

returns = ["/usr/openwin/bin", /usr/ucb", "/usr/bin", "/bin",
"/etc", "/usr/local/bin", "/usr/local/lib",
"/usr/shareware/bin", "/usr/shareware/lib", "."]
```

Problem 5 `(10 points): link_mapper(filename)`
Write a Python function named `link_mapper()` which takes a single string argument, named *filename*. *filename* will be the name of an HTML file. Check that the extension is correct, as in Problem 3. The function should return None if the extension is not ".html" or ".htm".

Create a dictionary. Create an entry in the dictionary whose key is *filename* and whose value is an empty list.

Read in the HTML file and then use regular expressions to capture every hyperlink in the file. For each hyperlink, you should create a tuple of the link text and the link address (URL).

For example, suppose the HTML file contains the following hyperlink definition:

```
   <a href="https://www.google.com">Search at Google</a>
```

You will create the following tuple:

```
   ("Search at Google", "https://www.google.com")
```

This tuple should then be appended to the list stored in the dictionary whose key is *filename*. After processing the entire contents of the HTML file, the function should return the dictionary.

It should be easy to see how this function could be extended to start building a record of links for many pages and start investigating the connections between them (Create a matrix and then draw a graph of the connections where each page is a node and each hyperlink is an edge). I am not asking you to do that, but you should appreciate how easily you can write a useful tool for internet investigation.

Problem 6 `(20 points): grammarly(text)`

Write a Python function called grammarly() that takes a single string argument, named *text*. The function will examine the contents of the string *text* for several different grammar mistakes. When a mistake is found, the mistake will then be corrected. That is, the function will progressively build new strings in which each grammar mistake is eliminated. The function will then return the corrected string.

There are many weird corner-cases for this question. Do your best; don't drive yourselves crazy trying to cover every possibility.

The grammar mistakes:

1. "i" instead of "I"
2. Uncapitalized beginning of a sentence.
3. Using "a" when you should use "an"
4. Repeated word
5. Missing Oxford Comma ("Our favorite companions are Garrus, Wrex and Tali." should be "Our favorite companions are Garrus, Wrex, and Tali.")
6. Unclosed parentheses.