Practice Project: Updating the CodeSkulptor Docs

This project is the designed to give you more substantial experience in working with strings and files. You are encouraged to complete this **optional** practice project before you attempt the required project for this week. Remember that, if you are struggling, you are welcome to examine our solution for the practice project provided at the end of the description.

Practice Project: Updating the CodeSkulptor Docs

As part of upgrading CodeSkulptor from Python 2 to Python 3, the original [CodeSkulptor Docs](http://www.codeskulptor.org/docs.html) (for Python 2) needed to be updated to reflect the language differences between Python 2 and Python 3. The resulting [CodeSkulptor3 Docs](http://py3.codeskulptor.org/docs.html#tabs-Python) consist of more than 10,000 lines of HTML. (Note the actual HTML for the Docs was generated via program from a much better organized PHP file that was created by hand.)

The most extensive part of this update involved modifying the 600+ code examples using print to reflect the fact that print is a function in Python 3. Making this update by hand would have been both time consuming and error prone. In practice project, you will write a script in Python 3 that automatically updates the HTML for the CodeSkulptor Docs such that almost all of the examples use correct Python 3 syntax for print (i.e. the arguments are enclosed in parentheses).

The structure of the HTML used in the CodeSkulptor Docs

The starting point for many scripting tasks is to examine the input data and understand its structure. We suggest that you begin by examining the Python 2 version of the Docs in a text editor such as Notepad. You will note that an HTML file is simply a text file organized by nested pairs of tags of the form <foobar>...</foobar>. Once this structure becomes apparent, automating the update of the example code involving print is surprisingly straightforward.

In designing our script, we need to identify an HTML tag that encapsulates the code examples using print. After closer examination, we note that the example code is always enclosed by by the HTML tag <pre> (meaning preformatted) so that the example is formatted to display as code. This observation suggest that we focus on extracting the <pre> blocks from the raw HTML and update any code examples that involve print in these blocks while leaving the containing text unmodified.

Recommended structure for your program

Based on the analysis in the previous section, we recommend that your script consist of three Python functions that update increasingly large logical components of the HTML file (a line, a <pre> block, and an entire file). The implementation of each of these functions has a similar three part structure:

1. The function splits the input string into smaller components,
2. The function then updates a subset of these smaller components using the update function for that component,
3. Finally, the function reassembles the modified components and returns the updated string.

For step two above, the update at the simplest level (a line) can be done directly using basic string operations. With this logic in mind. we suggest that you implement the three functions in this provided [starter code](https://storage.googleapis.com/codeskulptor-examples3/examples3_update_print_template.py):

* update\_line(line) - Takes a string line corresponding to a single line of Python 2 code. If the line corresponds to a print statement, the function updates the string such that the arguments to print are enclosed in parentheses. Note that any white space corresponds to indentation should be preserved. If the line does not correspond to a print statement, the line is left unchanged.
* update\_pre\_block(pre\_block) - Takes a string pre\_block corresponding to code example enclosed by a pair of <pre>...</pre> tags and returns a version of the text with the syntax of any enclosed print statements updated from Python 2 to Python 3. Note this function should call update\_line().
* update\_file(input\_file\_name, output\_file\_name) - Takes strings input\_file\_name  and output\_file\_name  corresponding to two file names as input. The function should open and read the HTML file corresponding to input\_file\_name , update the syntax of any examples of print statements in this file, and write result to the file corresponding to output\_file\_name . Note this function should call update\_pre\_block().

Note that the link to the starter code displays the file in your browser instead of loading the code into CodeSkulptor3. You will need to implement this practice project using desktop Python since CodeSkulptor3 can not access your local file system for security reasons. You can save the start code to your local file system by right-clicking and selecting "Save as ...".

You are **not** required to use our design and starter code since this project is not machine graded. If you want more of a challenge and wish to design your own solution, feel free to start from scratch. For required projects that are machine graded, you will need to follow our design to facilitate reliable grading.

Testing your script

We strongly recommend that you implement and test each function in the order specified above (smallest to largest). For example, do **not** attempt to implement update\_pre\_block() until you are confident that your implementation of update\_line() is correct. For each function, we have provided several tests to get you start in testing your code. We strongly recommend that you create more tests beyond our existing tests to increase your confidence in the correctness of your solution.

For update\_file(), the following test files are available: [table.html](https://storage.googleapis.com/codeskulptor-isp/course2/table.html) and [docs.html](https://storage.googleapis.com/codeskulptor-isp/course2/docs.html). You can open these files using a tool like Notepad to see the raw text as well as use a web-browser to view the formatted HTML. (Clicking on the lower row of tabs in "docs.html" displays the various components of the documentation.) Running the provided starter code with your solution on these examples produces two updated files, "table\_updated.html" and "docs\_updated.html".

Next, you should compare the updated HTML generated by your solution with the updated HTML generated by our solution code. Our updated files are available here: [table\_updated\_solution.html](https://storage.googleapis.com/codeskulptor-isp/course2/table_updated_solution.html) and [docs\_updated\_solution.html](https://storage.googleapis.com/codeskulptor-isp/course2/docs_updated_solution.html). Since determining the correctness of your updated version of the Docs is impossible to do by eye, the provided starter code imports [this code](https://storage.googleapis.com/codeskulptor-examples3/examples3_file_diff.py) that compares two specified files and reports where the first location where the two files differ. To use this code to test update\_file(), download this code into the same directory as your sources code and save it with the name "examples3\_file\_diff.py". You can then compare your updated HTML with our updated HTML solutions.

Once you have implemented and tested your implementation, you are welcome to examine [our solution](https://storage.googleapis.com/codeskulptor-examples3/examples3_update_print_solution.py) to the project that is available here.

Hints for the project

* Write a version of the each function that splits the input text into appropriate pieces and then reassembles the pieces to return the original text. (We suggest that you use the split() method here.) Once your splitting code is working, modify pieces of the split text appropriately.
* Note that our solution code does not update every instance of print in docs.html correctly. In particular, the example code for [bitwise operations](http://www.codeskulptor.org/docs.html#bitwise) in the Docs includes comments in the whitespace after the print statement. The updated version of these examples incorrectly includes the comment as an argument to print(). You are welcome to update your script to handle this case (or you can simply edit these examples by hand as we did).