Project Description: File Differences

In many scripting tasks you will need to process files. You often write programs that process different user inputs from files. It is also convenient to store configuration information for your programs in files so that the user does not have to specify the configuration over and over again. When files contain textual information, you will store the contents as strings within your programs and perform whatever string manipulation you need to do in order to accomplish the task at hand.

In this project you will find differences in the contents of two files. In particular, you will find the location of the first character that differs between two input files. You might want to do something like this if you are comparing how something has changed. It is convenient to present to the user the first difference to allow them to see what has happened. This could form the basis of a larger program that could find all of the differences between two files. For example, if you were to expand the program, you might then want to find the next difference after the user has read the first difference.

Preliminaries: Working on the Project

Coding Style

In this class, you will be asked to strictly follow a set of coding style guidelines. Good programmers not only get their code to work, but they also write it in a way that enables others to easily read and understand their code. Please read the style guidelines carefully and get into the habit of following them right from the start. A portion of your grade on the project will be based upon coding style.

Testing

You should always test your code as you write it. Do not try to solve the entire project before running it! If you do this, you will have lots of errors that interact in unexpected ways making your program very hard to debug. Instead, as you write each function, make sure you test it to ensure that it is working properly before moving on to the next function.

Throughout this course, we will be using a machine grader (OwlTest) to help you assess your code. You can submit your code to this OwlTest page to receive a preliminary grade and feedback on your project. The OwlTest page has a pale yellow background and does **not** submit your project to Coursera. OwlTest is just meant to allow you to test your project automatically. Note that trying to debug your project using the tests in OwlTest can be very tedious since they are slow and give limited feedback. Instead, we strongly suggest that you first test your program using your own tests. Also, note that each OwlTest link is specific to a particular project. You need to come back to this page and click the link above to ensure that you are running the tests for this project.

When you are ready to submit your code to be graded formally, submit your code to the assignment page for this project. You will be prompted to open a tool which will take you to the Coursera LTITest page. Note that the Coursera LTITest page looks similar to the OwlTest page, but they are not the same! The Coursera(!) Test page has a **white** background and does submit your grade to Coursera.

Note: Due to recent changes in Google App Engine, we have updated the machine grader for this course. If you experience any unexpected issues with OwlTest or CourseraTest. (e.g. the grader crashes or doesn't record your grade), feel free to email *interactivepython@rice.edu* with a short description of your issue and a copy of your code.

Project: File Differences

The project consists of multiple problems. Each problem will utilize functions you wrote for the previous problems, so we strongly suggest that implement and test these functions in the given order. We have provided the following template that you can use to get you started. It includes the signatures (name, parameters, and docstrings) for all of the functions that you will need to write. The code however, simply returns some arbitrary value no matter what the inputs are, so you will need to modify the body of the function to work correctly. You should not change the signature of any of the functions in the template, but you may add any code that you need to. You can also download all of the files used by OwlTest when testing your code as a zip file.

Note that the link to the template code displays the file in your browser instead of loading the code into CodeSkulptor3. **You will need to implement this 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...".

Problem 1: Finding the first difference between two lines

First, you will write a function called `singleline_diff` that takes two single line strings. You may assume that both strings are always a single line and do not contain any newline characters. The function should return the index of the first character that differs between the two lines. If the lines are the same, the function should return the constant `IDENTICAL`, which is already defined to be -1 in the provided template file.

If the lines are different lengths, but the entire shorter line matches the beginning of the longer line, then the first difference is located at the index that is one past the last character in the shorter line. In other words, no character after the end of the shorter line is defined to be different than whatever character exists in the longer line at that location.

Here is the signature of the `singleline_diff` function:

```
def singleline_diff(line1, line2):
  """
  Inputs:
    line1 - first single line string
    line2 - second single line string
  Output:
    Returns the index where the first difference between
    line1 and line2 occurs.

    Returns IDENTICAL if the two lines are the same.
  """
```

You need to write the code that implements this function.

Hints:

1. You do not need to check whether or not the two inputs are a single line or not. You may assume that they are.
2. You should first check the lengths of the two inputs and determine the length of the shorter line.
3. Look for differences in the lines up to the last character in the shorter line.
4. If you do not find any differences, think about what you should do in the two possible cases: (1) the lines are the same length and (2) the lines are different lengths.

Problem 2: Presenting the differences between two lines in a nicely formatted way

Next, you will write a function called `singleline_diff_format` that takes two single line strings and the index of the first difference and will generate a formatted string that will allow a user to clearly see where the first difference between two lines is located. A user is likely going to want to see where the difference is in the context of the lines, not just see a number. Your function will return a three line string that looks as follows:

```
abcd
==^
abef
```

The format of these three lines is:

1. The complete first line.
2. A separator line that consists of repeated equal signs ("=") up until the first difference. A "^" symbol indicates the position of the first difference.
3. The complete second line.

If either line contains a newline or carriage return character ("\n" or "\r") then the function returns an empty string (since the lines are not single lines and the output format will not make sense to a person reading it).

If the index is not a valid index that could indicate the position of the first difference of the two input lines, the function should also return an empty string (again because the output would not make sense otherwise). It must therefore be between 0 and the length of the shorter line. Note that you do **not** need to check whether the index actually identifies the correct location of the first difference, as that should have been computed correctly prior to calling this function.

Here is the signature of the `singleline_diff_format` function:

```
def singleline_diff_format(line1, line2, idx):
  """
  Inputs:
    line1 - first single line string
    line2 - second single line string
    idx   - index at which to indicate difference
  Output:
    Returns a three line formatted string showing the location
    of the first difference between line1 and line2.

    If either input line contains a newline or carriage return,
    then returns an empty string.

    If idx is not a valid index, then returns an empty string.
  """
```

You need to write the code that implements this function.

Hints:

1. Unlike the previous function, you do need to check whether the inputs are a single line this time.
2. Think about the string operations that you have learned about when constructing the separator line. Python provides mechanisms that allow you to do this relatively easily.
3. Make sure you add the appropriate newline characters to your output so that it is formatted correctly. In particular, there needs to be a newline at the end of each of the three lines.
4. Remember that this function is not checking whether or not the input index correctly identifies where the first difference is, it is simply displaying the difference assuming that it does. It is the job of `singleline_diff` to correctly compute the index.

Problem 3: Finding the first difference across multiple lines

Next, you will write a function called `multiline_diff` that takes two lists of single line strings. You may assume that the strings within the lists are all single lines. The function returns a tuple that indicates the line and index within that line where the first difference between the two lists occurs. If the contents of the two lists are the same, the function should return the tuple `(IDENTICAL, IDENTICAL)`. (Recall that the constant `IDENTICAL` is already defined to be -1 in the provided template file.)

The definition of whether two lines are the same or different and the resulting index for location of a difference is the same as it was for `singleline_diff`.

The line on which the first difference occurs should be the index into the input lists that correspond to that line. If the input lists are different lengths, but the entire shorter list matches the beginning of the longer list, the first difference is located at the line that is one past the last line in the shorter list (at index 0. In other words, no character on the line after the end of the shorter list is defined to be different than whatever character exists on that line in the longer list.

Here is the signature of the `multiline_diff` function:

```
def multiline_diff(lines1, lines2):
  """
  Inputs:
    lines1 - list of single line strings
    lines2 - list of single line strings
  Output:
    Returns a tuple containing the line number (starting from 0) and
    the index in that line where the first difference between lines1
    and lines2 occurs.

    Returns (IDENTICAL, IDENTICAL) if the two lists are the same.
  """
```

You need to write the code that implements this function.

Hints:

1. You do not need to check that each string in the input lists are single lines.
2. Do not forget to make use of any function(s) you have already written!
3. Notice the similarity between this function and `singleline_diff`. Make sure you think carefully about how to draw upon lessons you already learned when implementing that function.

Problem 4: Getting lines from a file

Next, you will write a function called `get_file_lines` that takes a filename as input. You may assume that the input names a file that exists and is readable. The function returns a list of single line strings, where each element of the list is one line from the file. The strings within the returned list should not contain any newline or carriage return ("\n" or "\r") characters.

Here is the signature of the `get_file_lines` function:

```
def get_file_lines(filename):
  """
  Inputs:
    filename - name of file to read
  Output:
    Returns a list of lines from the file named filename.  Each
    line will be a single line string with no newline ('\n') or
    return ('\r') characters.

    If the file does not exist or is not readable, then the
    behavior of this function is undefined.
  """
```

You need to write the code that implements this function.

Hints:

1. You do not need to check that the file exists or is readable. You may assume both.
2. Note that the Python functions for reading lines from files do not remove any line ending characters. You will need to do this yourself. Think about how you would find and remove them using what you have learned.
3. Always remember to close files when you are done using them!

Problem 5: Finding and formatting the first difference between two files

Finally, you will write a function called `file_diff_format` that takes two filenames as input. You may assume that the inputs name files that exist and are readable. The function returns a formatted string that will allow a user to clearly see where the first difference between two files is located. Your function will return a four line string that looks as follows:

```
Line 3:
abcd
==^
abef
```

The format of these four lines is:

1. A line that indicates which line the first difference occurs on (Line XXX:) where "XXX" is the line number (starting from 0).
2. The complete line XXX from the first file
3. A separator line that consists of repeated equal signs ("=") up until the first difference. A "^" symbol indicates the position of the first difference.
4. The complete line XXX from the second file.

The format of the second through fourth lines should be as they are defined for `singleline_diff_format`. The location of the first difference between the files should be defined in the same way that it was for `multiline_diff`.

If the two files are identical, the function should instead return the string "No differences\n".

Here is the signature of the `file_diff_format` function:

```
def file_diff_format(filename1, filename2):
  """
  Inputs:
    filename1 - name of first file
    filename2 - name of second file
  Output:
    Returns a four line string showing the location of the first
    difference between the two files named by the inputs.

    If the files are identical, the function instead returns the
    string "No differences\n".

    If either file does not exist or is not readable, then the
    behavior of this function is undefined.
  """
```

You need to write the code that implements this function.

Hints:

1. Do not forget to make use of any function(s) you have already written!

Go to next item ✓ Completed