For Part 1, the first step was to read in a text file. A `with` statement allowed for the file to be opened, then automatically closed at the end of the process. The `read()` function then allowed for the contents of the file to be read into a string. This information was found from here: https://docs.python.org/3/tutorial/inputoutput.html.

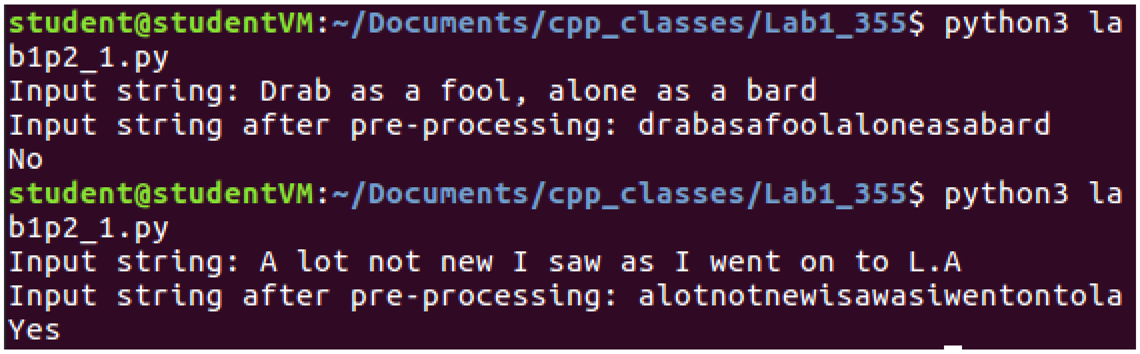
Next I had to do pre-processing to remove spaces, punctuation, and letter case. The first step was done using the `isalnum()` method for a string, found here: https://www.techiedelight.com/remove-non-alphanumeric-characters-string-python/. Lowercasing is easily done using the `lower()` method for a string, given in the introductory slides for this lab.

In order to check for a palindrome, all symmetric pairs of letters must be checked. This is done by using a for loop to check from the start and end of the word until the middle is reached. Since, for odd-length words, the middle does not have to be checked, the ending position to check for string `s` can be found using `len(s)//2`, which integer-divides the length of the string by 2.

A simple for loop iterates over the string until the ending limit, taking advantage of Python's ability to use negative indices to index a string in reverse. If a single mismatch is found, the string is not a palindrome, so the code exits, outputting `No`. If the code goes through all pairs without a mismatch, the string is a palindrome, so the code outputs `Yes`.

Part 1 code, executed with input of

a) Not a palindrome
b) Palindrome



```
student@studentVM:~/Documents/cpp_classes/Lab1_355$ python3 la
b1p2_1.py
Input string: Drab as a fool, alone as a bard
Input string after pre-processing: drabasafoolaloneasabard
No
student@studentVM:~/Documents/cpp_classes/Lab1_355$ python3 la
b1p2_1.py
Input string: A lot not new I saw as I went on to L.A
Input string after pre-processing: alotnotnewisawasiwentontola
Yes
```

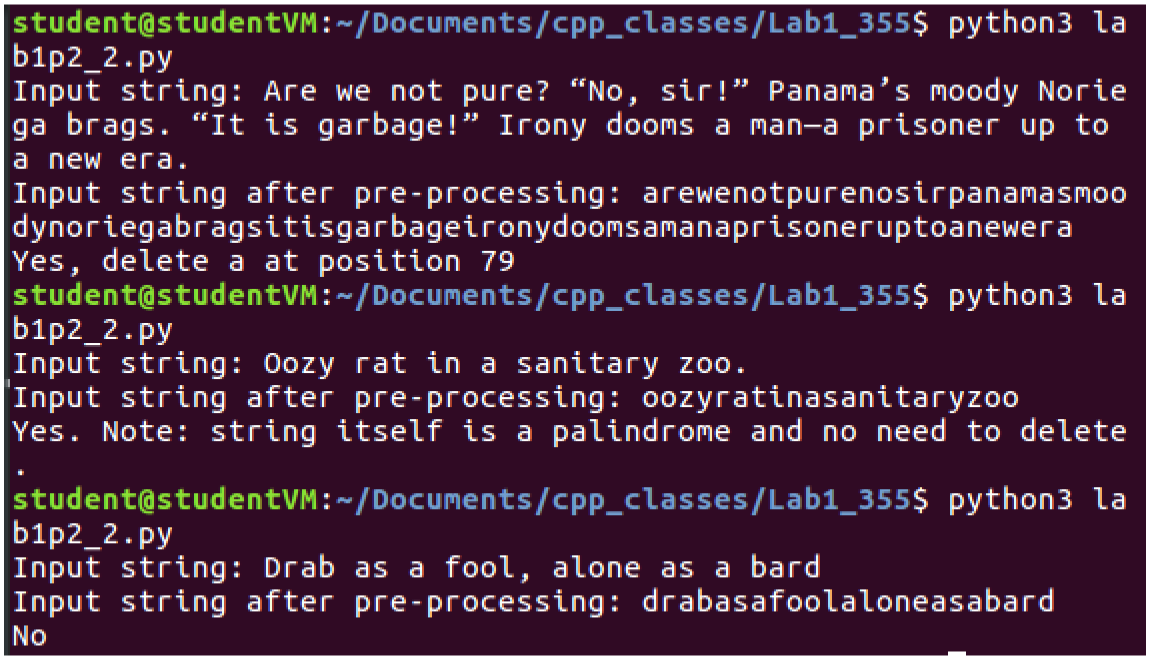
Part 2 uses all the same principles as Part 1, except with a few changes. A boolean, `error_found`, indicates whether the one allowable error has already been found, and variables `adj_start` and `adj_end` enable the for loop to skip over the faulty letter. When a mismatch is detected, the code checks whether this is the first mismatch, setting the `error_found` variable to `True` if so. If it detects that an error has already been found, the string does not satisfy the condition, and the program exits with a `No` output.

When an error is found, it is not clear whether the error is with the first half or the last half of the palindrome. The code must check whether skipping the letter in each half generates a valid pair. If either skip generates a valid pair, the program increments the respective `adj_` variable to skip the error character and continues from there. If neither pair matches, the string does not satisfy the condition, and the program exits with a `No` output.

After all pairs are checked successfully, the code outputs a `Yes`, with additional information about the faulty character and its position, if there is one. If the string is a palindrome with no faults, the program will output an additional note saying so.

Part 2 code, executed with input of

a) Almost-palindrome
b) Palindrome
c) Not a palindrome



```
student@studentVM:~/Documents/cpp_classes/Lab1_355$ python3 la
b1p2_2.py
Input string: Are we not pure? “No, sir!” Panama’s moody Norie
ga brags. “It is garbage!” Irony dooms a man—a prisoner up to
a new era.
Input string after pre-processing: arewenotpurenosirpanamasmoo
dynoriegabragsitisgarbageironydoomsamanaprisoneruptoanewera
Yes, delete a at position 79
student@studentVM:~/Documents/cpp_classes/Lab1_355$ python3 la
b1p2_2.py
Input string: Oozy rat in a sanitary zoo.
Input string after pre-processing: oozyratinasanitaryzoo
Yes. Note: string itself is a palindrome and no need to delete
.
student@studentVM:~/Documents/cpp_classes/Lab1_355$ python3 la
b1p2_2.py
Input string: Drab as a fool, alone as a bard
Input string after pre-processing: drabasafoolaloneasabard
No
```