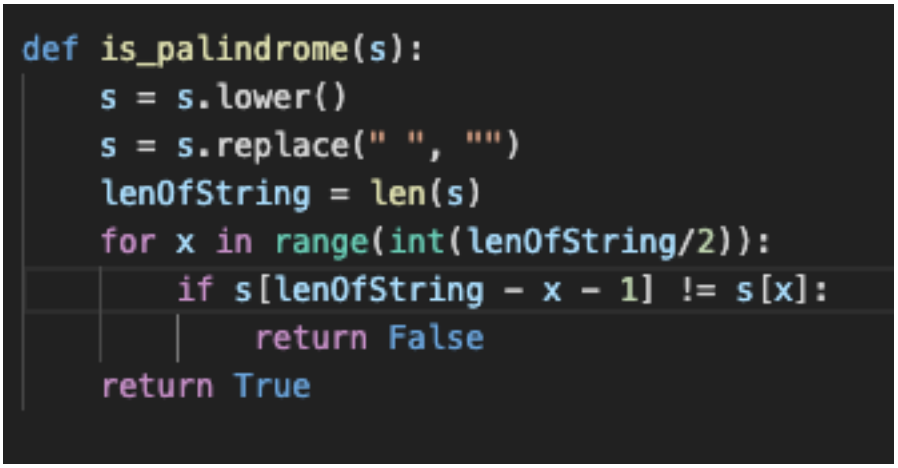
I would start by optimizing the string iteration to use a single loop. The single loops will iterate up to half the length of the string, having a time complexity of O(n/2). This should give boost in performance over the use of the nested loop present in sample in the implementation. This optimization would require a change to the conditional logic inside the loop to now compare the current index the loop is over, let's call it 'x', against the character at index 'len(s) - x - 1' which would be the mirroring character needed for comparison to check if we have a palindrome, thus the reason for only looping over half of the string.
From a readability point of view, I would probably clean up to check that if the characters being compared don't equal each other then return false which should allow to save a couple of lines of code and make it more straightforward to see what is expected from the function. Finally, the optimized function would look like the following



```
def is_palindrome(s):
    s = s.lower()
    s = s.replace(" ", "")
    lenOfString = len(s)
    for x in range(int(lenOfString/2)):
        if s[lenOfString - x - 1] != s[x]:
            return False
    return True
```