

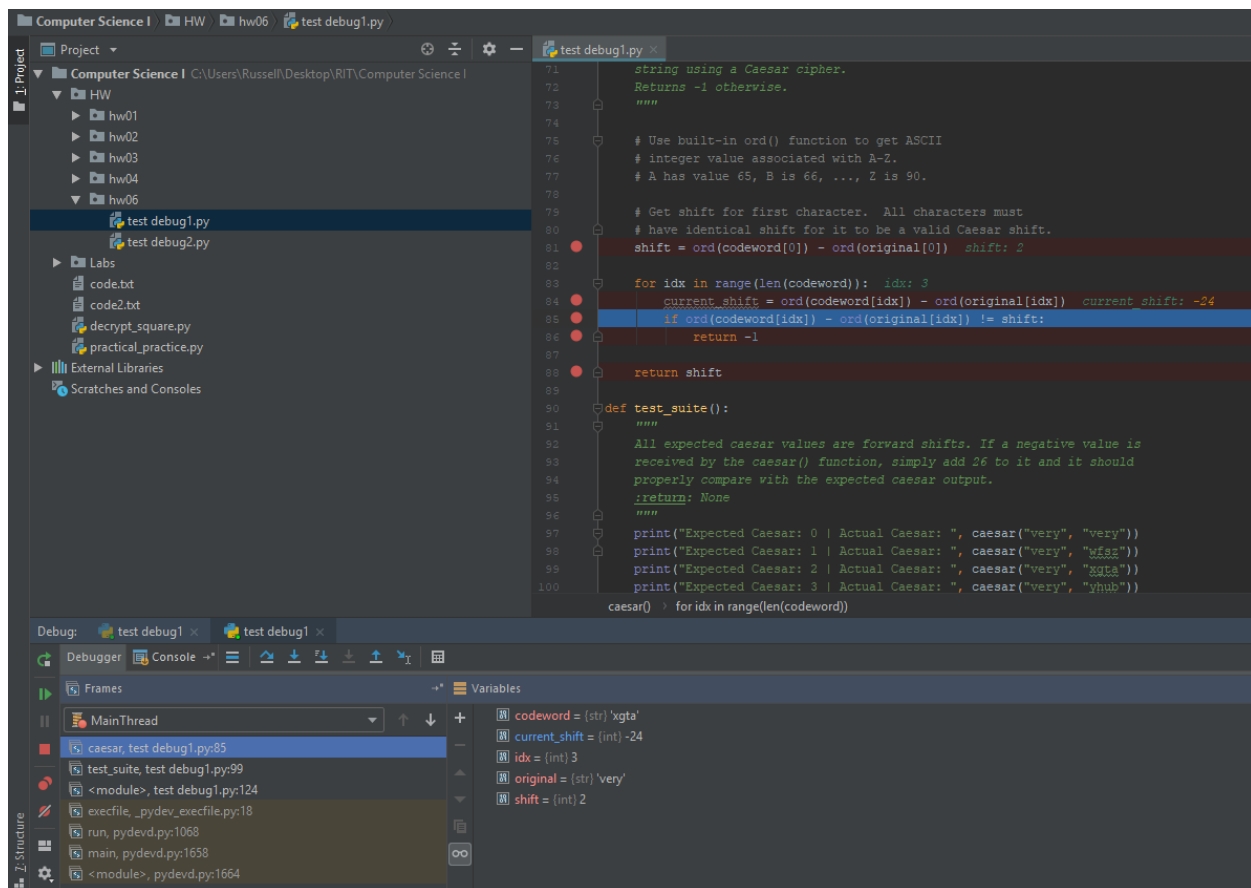
Russell Harvey
Professor Steele
CSCI-141
4 October 2018

Homework 6 – Debugging

Caesar:

```
a) print("Expected Caesar: 0 | Actual Caesar: ", caesar("very", "very"))  
    print("Expected Caesar: 1 | Actual Caesar: ", caesar("very", "wfsz"))  
  
Expected Caesar: 0 | Actual Caesar: 0  
Expected Caesar: 1 | Actual Caesar: 1
```

- b) For the first print statement, since the “shift” variable is 0, the program runs as it is supposed to because every ASCII value in the original and the codeword is the same. As for the second, since every character in the string is shifted up by one, the “shift” variable is properly assigned to 1 and the function will return 1. These work because in the first two working codeword/original pairs for the word very, none of the codeword characters’ ASCII values return to a lower ASCII value than the original’s.
- c) When the program encounters a character with a lower ASCII in the codeword than the original. The statement that assigns “shift” a value gets tossed out of whack. While shift is assigned the correct shift value, when the loop moves to a character that does not have a lower ASCII value than the original the comparison will not be equivalent to shift and will result in the function returning -1 instead of the proper shift value. To fix this, I implemented a logic system where if the ASCII value of the original is greater than the codeword, it would take the difference of the original minus the codeword instead of the opposite. It takes this difference and subtracts it from 26, assigning it to shift giving shift the correct value for the loop to work. I also added an additional few statements in the for loop that checks for this same issue. If it is present, the loop performs the same operation stated above within an if statement so that the shift value will match the shift value at every index of the string.



Longest Consecutive Matching Substring:

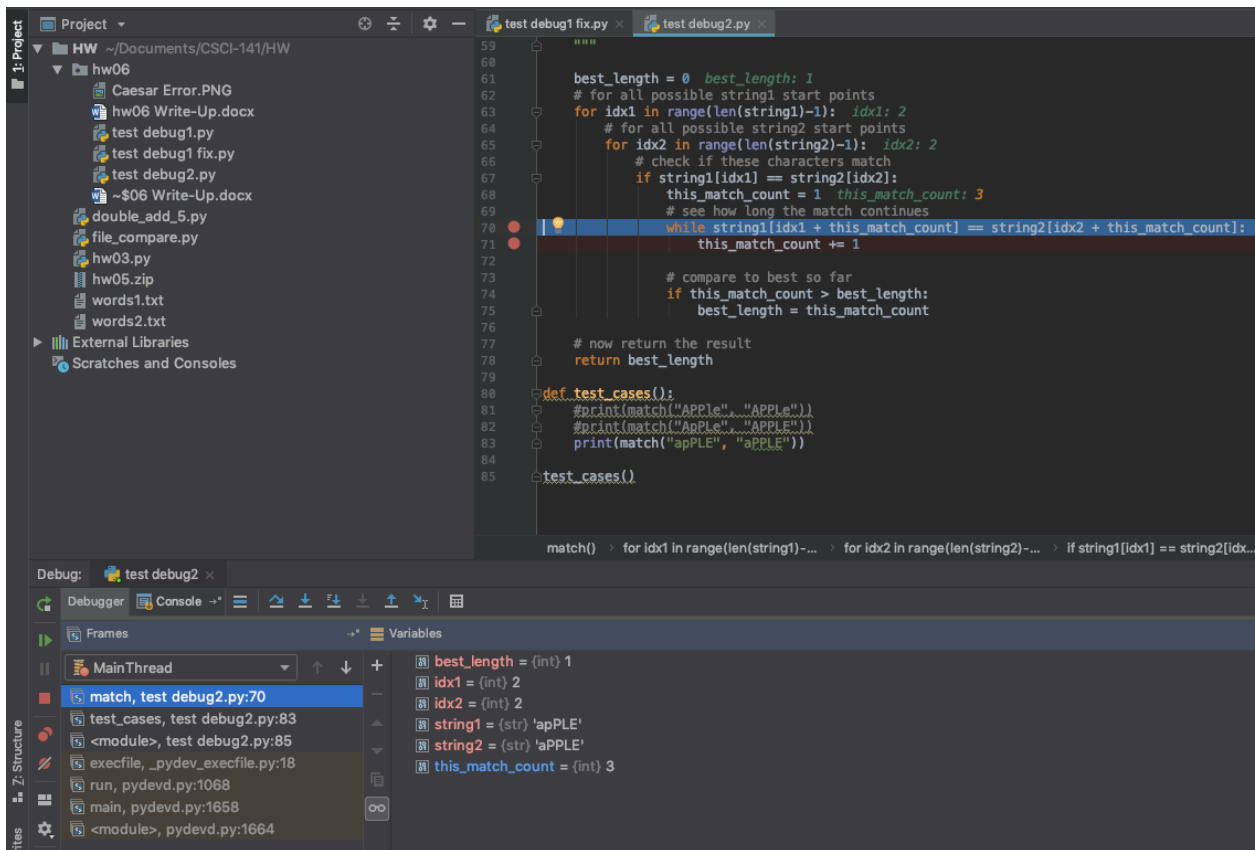
```

a) print("Expected Output: 3 |", " Actual Output: ", match("APpLe", "APpLe"))
   print("Expected Output: 2 |", " Actual Output: ", match("ApPLe", "APPLE"))

Expected Output: 3 | Actual Output: 3
Expected Output: 2 | Actual Output: 2
  
```

- The two examples above function because they don't ever go beyond the maximum index in each string. By the nature of the code if the matching substring goes to the end of the string, the while loop in the match function will eventually go out of range producing an error. These two examples do not go past the index range of the given strings which will cause the while loop to work properly allowing these functions to work. Also if the second string was any longer it would once again go out of the index range because the index range of the first string is smaller than the index range of the second.
- The error present within the code was that if the LCMS (longest consecutive matching substring) ended at the tail of one of the strings or if one string was longer than the other program would return an index error. This is because the while loop would iterate past the shorter string's index and cause it to break the while loop. To fix this, I created two variables called max_length1 and max_length2 and set them equal to the length of each respective string minus one. Then I changed the while loop to check if the current index

of the string was past or equal to max_length1 or 2 for the respective strings. I then put the conditions inside the old while statement into an if statement inside the while loop. The function now runs properly and returns proper amounts.



The screenshot shows a Python IDE with a project named 'HW' located at '~/.Documents/CSCI-141/HW'. The file explorer on the left lists various files including 'Caesar Error.PNG', 'hw06 Write-Up.docx', 'test debug1.py', 'test debug1 fix.py', 'test debug2.py', '\$06 Write-Up.docx', 'double_add_5.py', 'file_compare.py', 'hw03.py', 'hw05.zip', 'words1.txt', and 'words2.txt'. The code editor displays a Python script with the following content:

```
59 """
60
61 best_length = 0 best_length: 1
62 # for all possible string1 start points
63 for idx1 in range(len(string1)-1): idx1: 2
64 # for all possible string2 start points
65 for idx2 in range(len(string2)-1): idx2: 2
66 # check if these characters match
67 if string1[idx1] == string2[idx2]:
68     this_match_count = 1 this_match_count: 3
69     # see how long the match continues
70     while string1[idx1 + this_match_count] == string2[idx2 + this_match_count]:
71         this_match_count += 1
72
73     # compare to best so far
74     if this_match_count > best_length:
75         best_length = this_match_count
76
77 # now return the result
78 return best_length
79
80 def test_cases():
81     #print(match("APLe", "APLe"))
82     #print(match("ApLe", "APLe"))
83     print(match("apLE", "aPPLE"))
84
85 test_cases()
```

The debugger window at the bottom shows the current state of the program. The 'Frames' pane lists the call stack, with the current frame being 'match, test debug2.py:70'. The 'Variables' pane shows the following values:

- best_length = (int) 1
- idx1 = (int) 2
- idx2 = (int) 2
- string1 = (str) 'apLE'
- string2 = (str) 'aPPLE'
- this_match_count = (int) 3