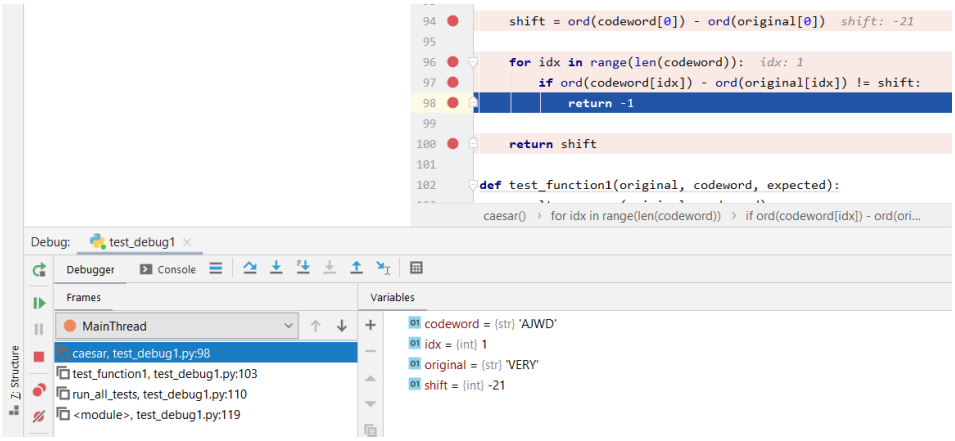
Tyler Padrao 10/5/19
CSC1 HW06

Caesar Shift:

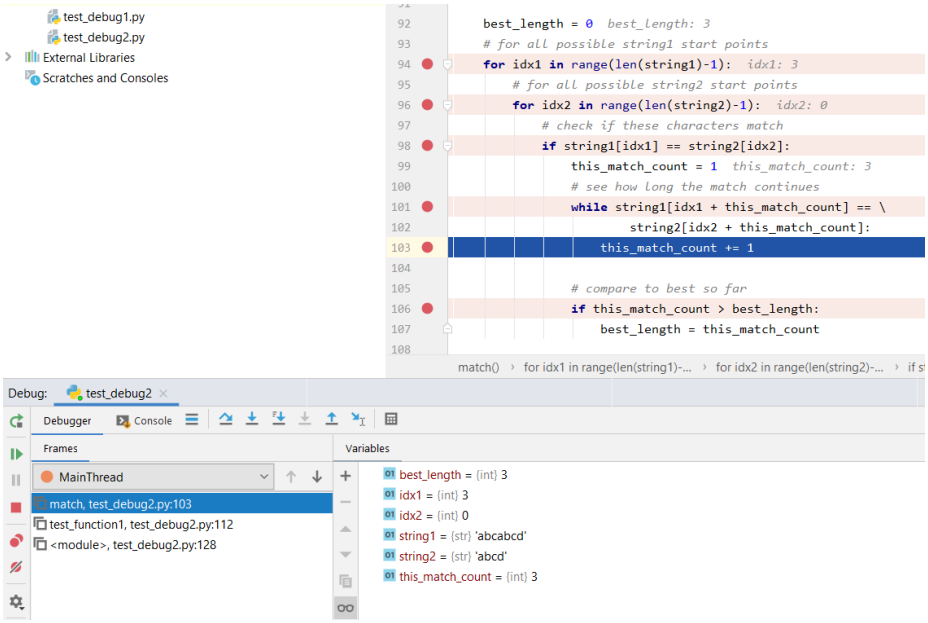


Examples of passing code:

test_function1("DOG", "JUM", 6)
test_function1("nope", "opqf", 1)

Within the Caesar shift program, these two test examples work fine even though there's flaws in the program. There are 26 characters in the alphabet. When the Caesar shift has a shift that exceeds the last letter of the alphabet, the shift wraps back over to the first letter again. The reason why these two test functions work is because the shift does not exceed the last letter of the alphabet. "DOG" has a shift of 6, resulting in "JUM". "nope" has a shift of 1, resulting in "opqf". These shifts for example never exceed the last letter of the alphabet, resulting in a pass. Thus, these test functions will pass without problems, since the shift never exceeds the last letter of the alphabet. The first bug in the program is that the shift is always equivalent to a negative number. The shift must be from 1-25. The program recognizes the correct shift but takes the negative shift since you're always cycling through one direction. The fix to make the shift positive is if the shift is negative, add 26 to the negative shift. Since there's 26 letters in the alphabet, for example, a shift of -21 is equal to 5. The second error is the condition within the next if statement is always going to be true, therefore always returning -1 if there is a negative shift. You would use the same idea, if the shift is negative, add 26 to make it positive.

Longest Consecutive matching substring:



Examples of passing code:

test_function1("established", "ballistic", 3)
test_function1("bookkeeper", "bookkeeping", 8)

Within the longest consecutive matching substring program, these two examples work fine even though there's flaws in the program. The objective of the program is to find the longest consecutive matching substring. The reason why the above examples work is because there's only one matching substring between the two strings. The matching substring for "established" and "ballistic" is "lis" between the two strings. Since there's only one matching substring, the program will still pass. The same idea for "bookkeeper" and "bookkeeping". The matching substring is "bookkeep". Since there's only one matching substring, the example will pass. The first error within the code is it has a problem identifying matching substrings for two of the same strings. The simple fix for this is before it gets into the for loop, check to see if the strings are equal and if they are simply return the length of the string. The second bug is that the program throwing an index out of bounds error due to the condition in the while loop. When there's two matching substrings such as "abcd" and "abcabcd", it will throw an error. The program is only comparing the index of the two strings. To fix this condition, you must check if the index plus the "this_match_count" incrementor to see if its less than the length of the string. Once you do this for both strings, you also must check if the indexes for string one and two are equal to each other. If all these conditions satisfy, then increment "this_match_count".