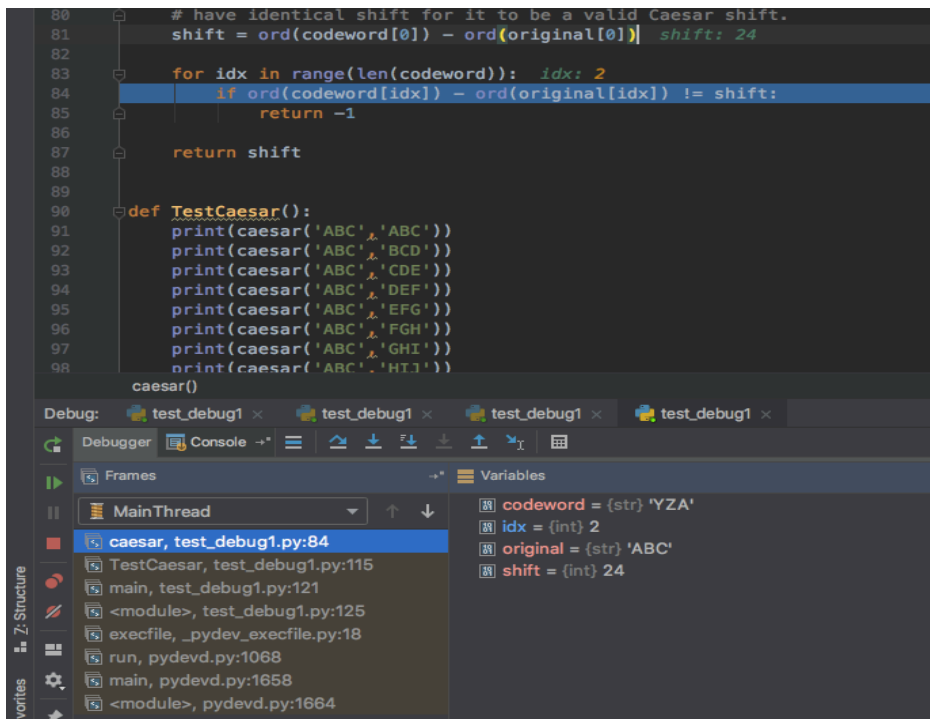


Youssef Naguib
CS1
Prof. Monika Polak
10/6/18

Problem 1: Caesar cipher

Bugs: The Caesar function works with test cases of shifts from 0 to 23, however when the shift is 24, 25, or 26, it does not work because the ASCII values repeat when the alphabet restarts. For example Caesar('ABC', 'YZA') is a shift of 24, yet $\text{Ord}(\text{A}) - \text{Ord}(\text{C})$ returns -2. The program therefor returns -1 when it should return 24.



```
80 # have identical shift for it to be a valid Caesar shift.
81 shift = ord(codeword[0]) - ord(original[0]) shift: 24
82
83 for idx in range(len(codeword)): idx: 2
84     if ord(codeword[idx]) - ord(original[idx]) != shift:
85         return -1
86
87 return shift
88
89
90 def TestCaesar():
91     print(caesar('ABC', 'ABC'))
92     print(caesar('ABC', 'BCD'))
93     print(caesar('ABC', 'CDE'))
94     print(caesar('ABC', 'DEF'))
95     print(caesar('ABC', 'EFG'))
96     print(caesar('ABC', 'FGH'))
97     print(caesar('ABC', 'GHI'))
98     print(caesar('ABC', 'HTI'))
99
100 caesar()
```

Debugger: test_debug1 x test_debug1 x test_debug1 x test_debug1 x

Debugger Console

Frames

- MainThread
- caesar, test_debug1.py:84
- TestCaesar, test_debug1.py:115
- main, test_debug1.py:121
- <module>, test_debug1.py:125
- execfile, _pydev_execfile.py:18
- run, pydevd.py:1068
- main, pydevd.py:1658
- <module>, pydevd.py:1664

Variables

- codeword = (str) 'YZA'
- idx = (int) 2
- original = (str) 'ABC'
- shift = (int) 24

Test Cases that are still correct with bug in place:

For shifts less than 23, this bug does not apply because the alphabet does not restart. This means that the $\text{Ord}(\text{codeword}[\text{idx}]) - \text{Ord}(\text{original}[\text{idx}])$ will always equal the shift. This is due to the fact that the ASCII values constantly increase by 1.

Example 1.

('ABC', 'ABC') returns 0 because the string does not shift. This is correct.

Example 2.

('ABC', 'MNO') returns 12 because each ASCII value for each character in 'codeword' minus each ASCII value for each character in 'original' is 12. The string shifts by 12.

Example 1.

```

88
89
90 def TestCaesar():
91     print(caesar('ABC', 'ABC'))
92
93
94 def main():
95     TestCaesar()
96
97 if __name__ == "__main__":
98     main()

```

Run: test_debug1 x /usr/local/bin/python3.7 "/Users/Youssef" 0
Process finished with exit code 0

Example 2.

```

88
89
90 def TestCaesar():
91     print(caesar('ABC', 'MNO'))
92
93
94
95 def main():
96     TestCaesar()
97
98
99
100 if __name__ == "__main__":
101     main()

```

Run: test_debug1 x /usr/local/bin/python3.7 "/Users/Youssef" 12
Process finished with exit code 0

Solution:

To fix this bug, I added a new variable named negShift. It is shown below.

$\text{Negshift} = \text{ord}(\text{original}[0]) - \text{ord}(\text{codeword}[0]) + 26$

I then added an elif statement, after checking if the ASCII value isn't equal to shift.

If it isn't equal to shift but is equal to negshift, then shift becomes negshift. Practically, negshift is returned.

If the ASCII value isn't equal to shift or negshift then -1 is returned.

The code to the solution is shown below. Test cases for each possible output are called in the file.

```

shift = ord(codeword[0]) - ord(original[0])
negShift = ord(original[0]) - ord(codeword[0]) + 26

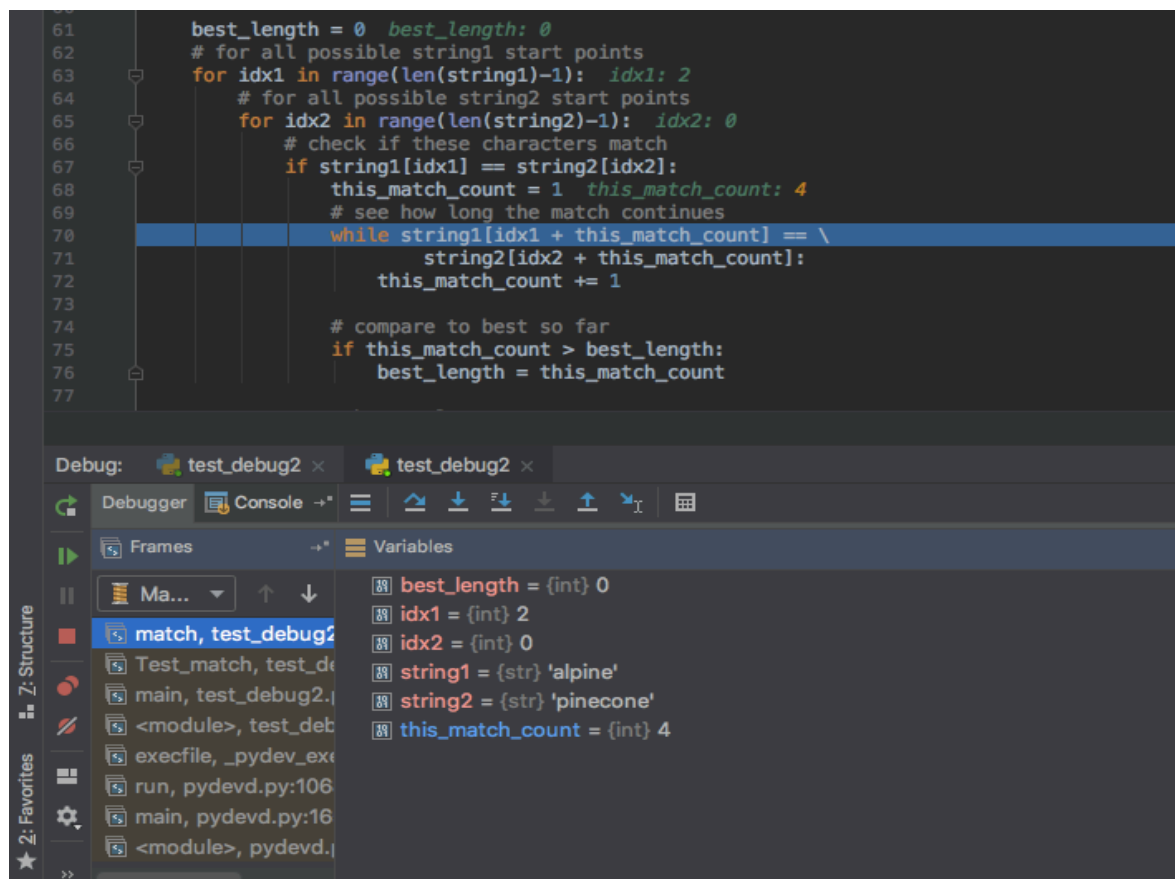
for idx in range(len(codeword)):
    if ord(codeword[idx]) - ord(original[idx]) != shift:
        return -1
    elif ord(original[idx]) - ord(codeword[idx]) + 26 != negShift:
        shift = shift + negshift

return shift

```

Problem 2: Longest Consecutive Matching Substring

Bugs: the function `Match(string1, string2)` returns an error when string 2 is longer than string 1. This is because `string1` runs out of indexes and cannot compare with string 2, which has more indexes. In the example `alpine` had an index of 5 while `pinecone` has an index of 7. The program should return 4, because both strings contain 'pine' which has a length of 4. However, the strings cannot be compared because they are not of equal length. After fixing this issue, another bug was discovered. If the identical part of the strings in `string2` was at the end of the string, and `string2` was longer than `string1`, the index would still go out of range because the length of `string2` would still be larger than the index plus the number of matching consecutive letters or numbers. For example `Match('newyork', 'brandnew')`, would return an error, instead of 3. Below is an example of the debugger right before the while loop checks an index out of range of string 1.



The screenshot shows a Python IDE with a debugger. The code is as follows:

```
61 best_length = 0
62 # for all possible string1 start points
63 for idx1 in range(len(string1)-1):
64     # for all possible string2 start points
65     for idx2 in range(len(string2)-1):
66         # check if these characters match
67         if string1[idx1] == string2[idx2]:
68             this_match_count = 1
69             # see how long the match continues
70             while string1[idx1 + this_match_count] == \
71                 string2[idx2 + this_match_count]:
72                 this_match_count += 1
73
74             # compare to best so far
75             if this_match_count > best_length:
76                 best_length = this_match_count
77
```

The debugger is open, showing the current state of the program. The **Variables** pane on the right displays the following values:

- `best_length = {int} 0`
- `idx1 = {int} 2`
- `idx2 = {int} 0`
- `string1 = {str} 'alpine'`
- `string2 = {str} 'pinecone'`
- `this_match_count = {int} 4`

The **Frames** pane on the left shows the call stack, with the current frame being `match, test_debug2`.

Test cases that are still correct with bug in place:

Below are three examples of the function running correctly with the bug in place. Example 1 compares two strings of equal length, with no identical or consecutive letters. 0 should be returned and 0 is returned. This is a valid answer. In example 2, `string1` is longer than `string2`, and there are 2 identical letters that are consecutive in both strings, these are 'ro'. This test case should return 2, and does return 2. This is a valid answer. The third example is similar to example 2, but there is only 1 identical letter in string 1 and 2. This happens to be an 'o'. The test

should return 1, and does return 1, so it is valid. The conclusion is that if string 1 and 2 are of equal length, or if string 1 is longer than string 2, the function will return the correct answer.

Example 1.

```
def Test_match():
    print(match('rit', 'abc'))

def main():
    Test_match()

if __name__ == "__main__":
    main()
```

test_debug2 x /usr/local/bin/python3.7 "/Users/You
0
Process finished with exit code 0

Example 2.

```
def Test_match():
    print(match('problem', 'rolex'))

def main():
    Test_match()

if __name__ == "__main__":
    main()
```

Test_match()
n: test_debug2 x /usr/local/bin/python3.7 "/Users/You
2
Process finished with exit code 0

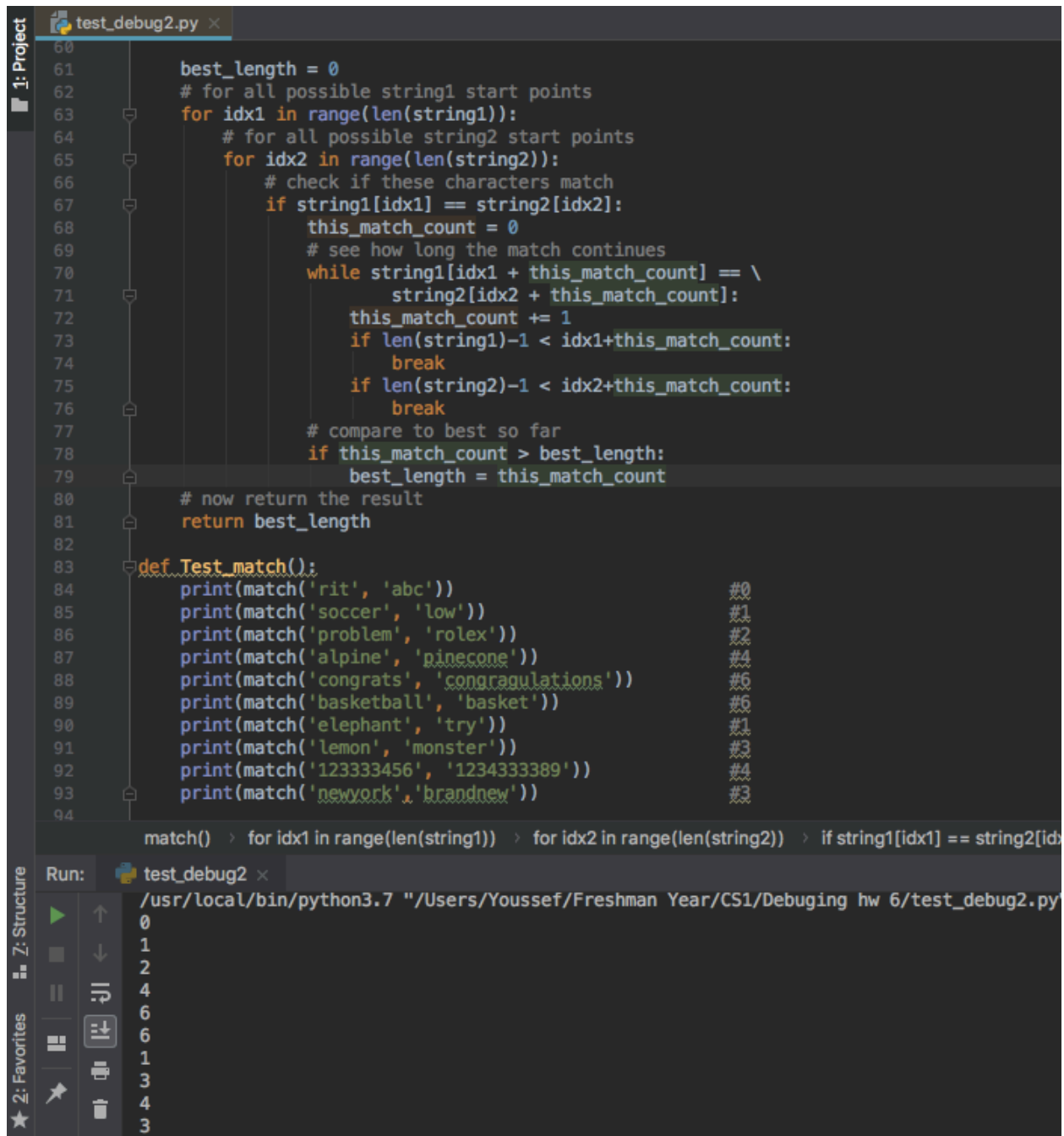
Example 3.

```
79 return best_length
80
81 def Test_match():
82     print(match('soccer', 'low'))
83
84
85 def main():
86     Test_match()
87
88 if __name__ == "__main__":
89     main()
90
```

Test_match()
Run: test_debug2 x /usr/local/bin/python3.7 "/Users/You
1
Process finished with exit code 0

Solution:

I made the `this_match_count` variable equal to 0 when the first identical letter is found, rather than equal to 1. I also added 2 condition in the while loop which states that if the `this_match_count` variable is longer than $(\text{string1})-1$ or $(\text{string2})-1$, break it of the loop. This makes sure the while loop doesn't keep checking indexes that are out of range of the string. The correct code is shown below, and with it test cases the code working correctly with `string2` being longer than `string1`, `string 1` being longer than `string2`, and both `string1` and `string2` being equal length. The correct values are returned with no errors.



```
60
61 best_length = 0
62 # for all possible string1 start points
63 for idx1 in range(len(string1)):
64     # for all possible string2 start points
65     for idx2 in range(len(string2)):
66         # check if these characters match
67         if string1[idx1] == string2[idx2]:
68             this_match_count = 0
69             # see how long the match continues
70             while string1[idx1 + this_match_count] == \
71                 string2[idx2 + this_match_count]:
72                 this_match_count += 1
73                 if len(string1)-1 < idx1+this_match_count:
74                     break
75                 if len(string2)-1 < idx2+this_match_count:
76                     break
77             # compare to best so far
78             if this_match_count > best_length:
79                 best_length = this_match_count
80
81 # now return the result
82 return best_length
83
84 def Test_match():
85     print(match('rit', 'abc'))           #0
86     print(match('soccer', 'low'))        #1
87     print(match('problem', 'rolex'))     #2
88     print(match('alpine', 'pinecone'))   #4
89     print(match('congrats', 'congragulations')) #6
90     print(match('basketball', 'basket')) #6
91     print(match('elephant', 'try'))       #1
92     print(match('lemon', 'monster'))     #3
93     print(match('123333456', '1234333389')) #4
94     print(match('newyork', 'brandnew'))   #3
```

match() > for idx1 in range(len(string1)) > for idx2 in range(len(string2)) > if string1[idx1] == string2[idx2]

Run: test_debug2 ×
/usr/local/bin/python3.7 "/Users/Youssef/Freshman Year/CS1/Debuging hw 6/test_debug2.py"

0
1
2
4
6
6
1
3
4
3