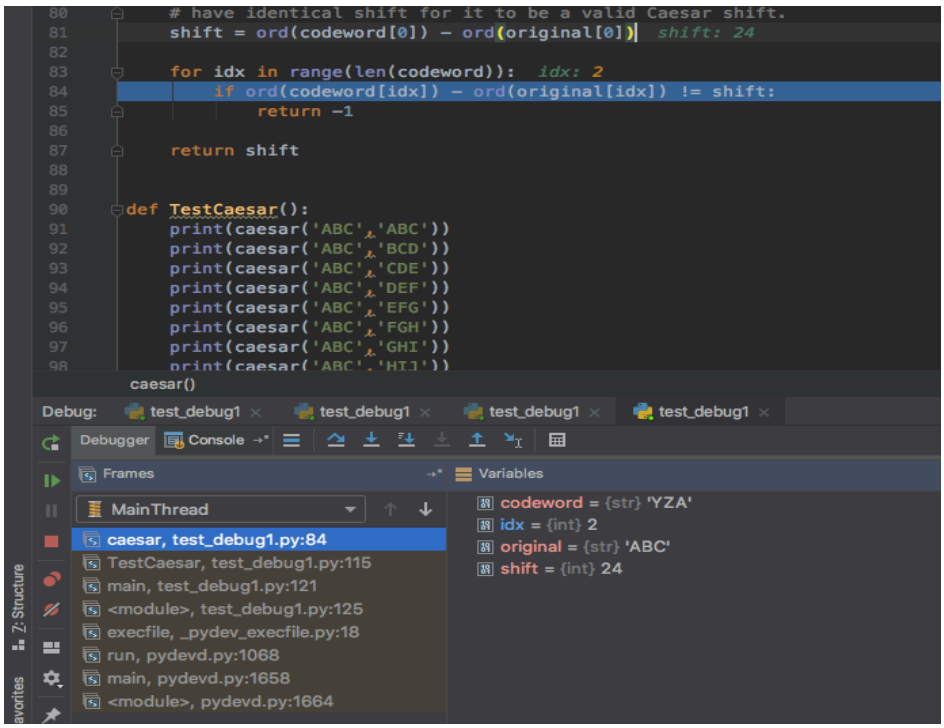
Youssef Naguib
CS1
Prof. Monika Polak
10/6/18

Problem 1: Caesar cipher

Bugs: The Ceaser 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 Ceaser('ABC', 'YZA') is a shift of 24, yet Ord(A) – Ord(C) returns -2. The program therefor returns -1 when it should return 24.



Test Cases that are still correct with bug in place:
For shifts less than 23, this bug does not apply because because the alphabet does not restart. This means that the Ord(codeword[idx]) – Ord(original[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 'cordword ' minus each ASCII value for each character in 'original' is 12. The string shifts by 12.

Example 1.

```
def TestCaesar():
    print(caesar('ABC','ABC'))


def main():
    TestCaesar()

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

```
Run: test_debug1
/usr/local/bin/python3.7 "/Users/Yous
0

Process finished with exit code 0
```

Example 2.

```
def TestCaesar():
    print(caesar('ABC','MNO'))



def main():
    TestCaesar()


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

```
TestCaesar()
Run: test_debug1
/usr/local/bin/python3.7 "/Users/Yousse
12

Process finished with exit code 0
```

Solution:
To fix this bug, I added a new variable named negShift. It is shown below.
Negshift = ord(original[0]) - ord(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.

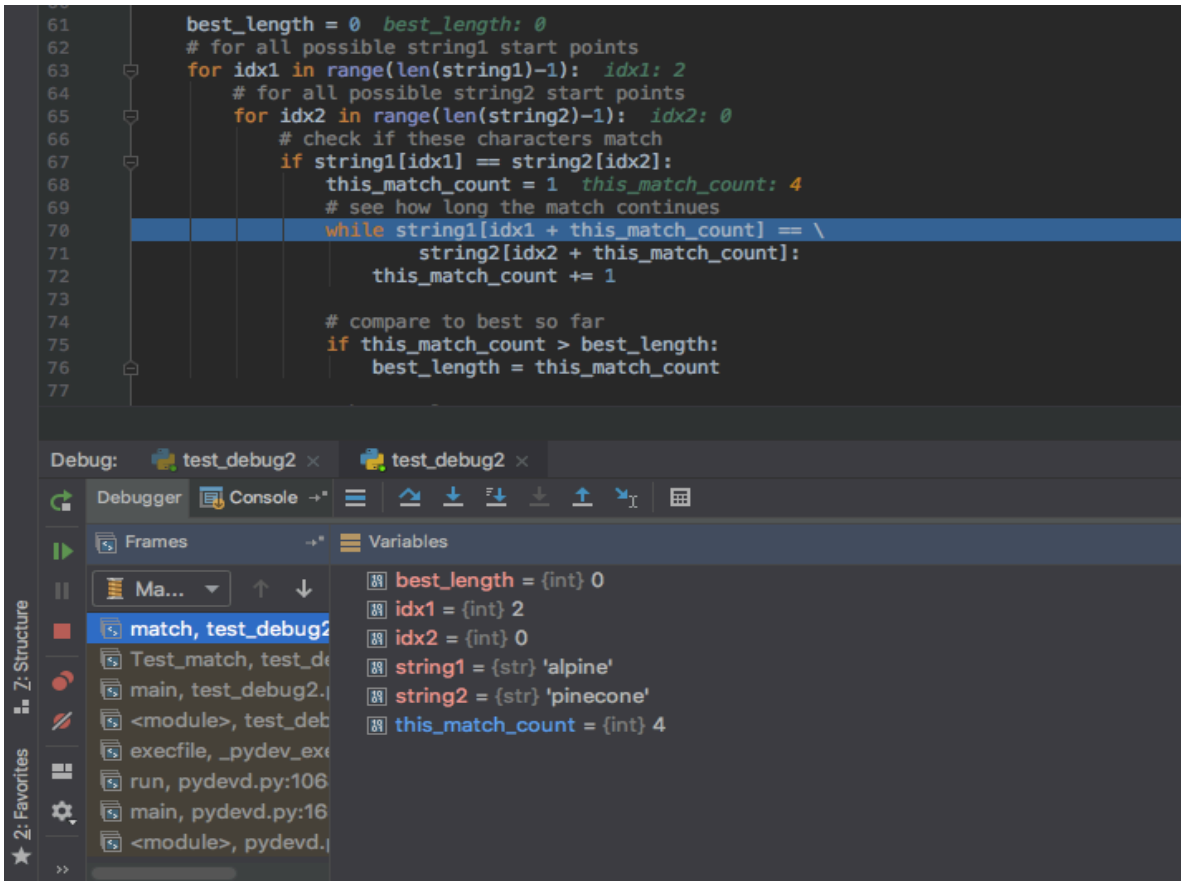
```
shift = ord(codeword[0]) - ord(original[0])
negShift = ord(original[0]) - ord(codeword[0]) + 26

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 string 2 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.



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 string 2, 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 1and 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
/usr/local/bin/python3.7 "/Users/Yous
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
/usr/local/bin/python3.7 "/Users/You
2

Process finished with exit code 0
```

Example 3.

```
def Test_match():
    print(match('soccer', 'low'))


def main():
    Test_match()

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

```
Test_match()
Run:  test_debug2
/usr/local/bin/python3.7 "/Users/Yo
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 (string1)-1 or (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.

```python
    best_length = 0
    # for all possible string1 start points
    for idx1 in range(len(string1)):
        # for all possible string2 start points
        for idx2 in range(len(string2)):
            # check if these characters match
            if string1[idx1] == string2[idx2]:
                this_match_count = 0
                # see how long the match continues
                while string1[idx1 + this_match_count] == \
                        string2[idx2 + this_match_count]:
                    this_match_count += 1
                    if len(string1)-1 < idx1+this_match_count:
                        break
                    if len(string2)-1 < idx2+this_match_count:
                        break
                # compare to best so far
                if this_match_count > best_length:
                    best_length = this_match_count
    # now return the result
    return best_length

def Test_match():
    print(match('rit', 'abc'))                          #0
    print(match('soccer', 'low'))                       #1
    print(match('problem', 'rolex'))                    #2
    print(match('alpine', 'pinecone'))                  #4
    print(match('congrats', 'congragulations'))         #6
    print(match('basketball', 'basket'))                #6
    print(match('elephant', 'try'))                     #1
    print(match('lemon', 'monster'))                    #3
    print(match('123333456', '1234333389'))             #4
    print(match('newyork','brandnew'))                  #3
```

```
/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
```