

Jarred Moyer

Computer Science

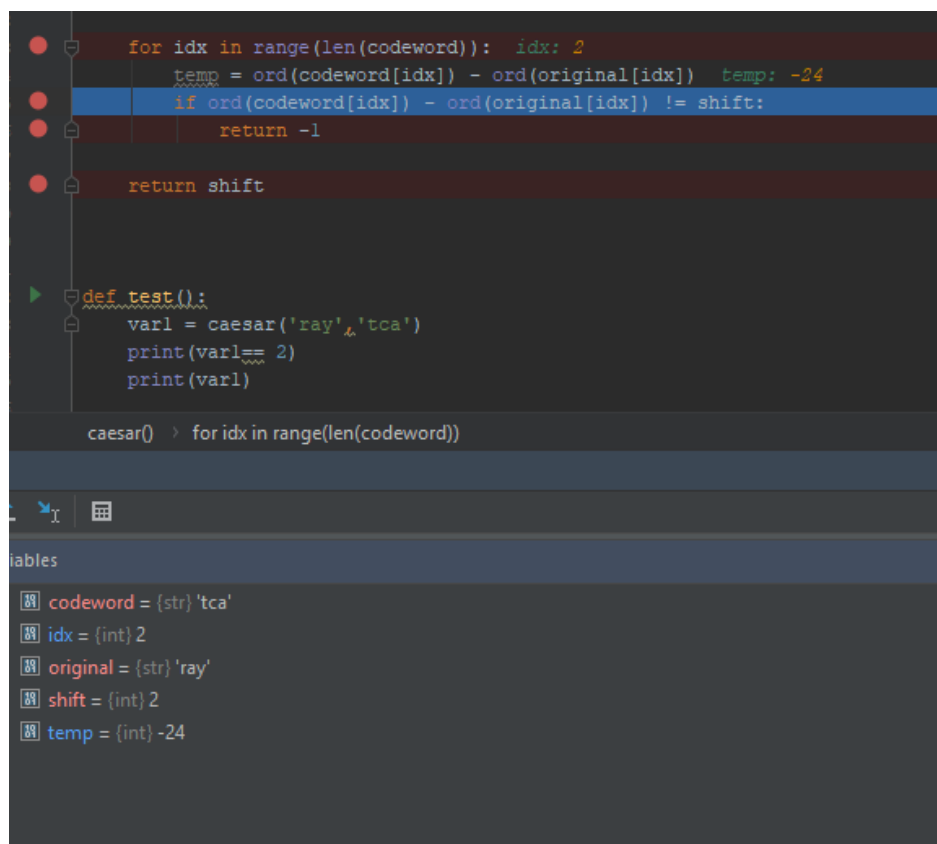
Homework 06

Professor Park

10/6/18

Problem 1:

Two false positive inputs for the Caesar function are 'abc','bde' and 'call','dbmm'. For both examples, no letter is encoded in past z. For example, in the first test case, the string is encoded with a shift of one. Given that no input character is with in one character of having to be looped back onto a, no error will be produced. An example of the error can be seen when encoding 'xyz' to 'abc'. A shift of three will map 'xyz' to 'abc'. However, due to all characters having to loop to the start of the alphabet, the program will produce a semantic error. These errors can be fixed by making 'a', the letter after 'z' and so on. This will attach the bottom of the alphabet, to the top of the alphabet. Allowing the cipher to shift around a circle rather than a straight line.



```
for idx in range(len(codeword)): idx: 2
    temp = ord(codeword[idx]) - ord(original[idx]) temp: -24
    if ord(codeword[idx]) - ord(original[idx]) != shift:
        return -1

return shift

def test():
    var1 = caesar('ray', 2)
    print(var1)
    print(var1)
```

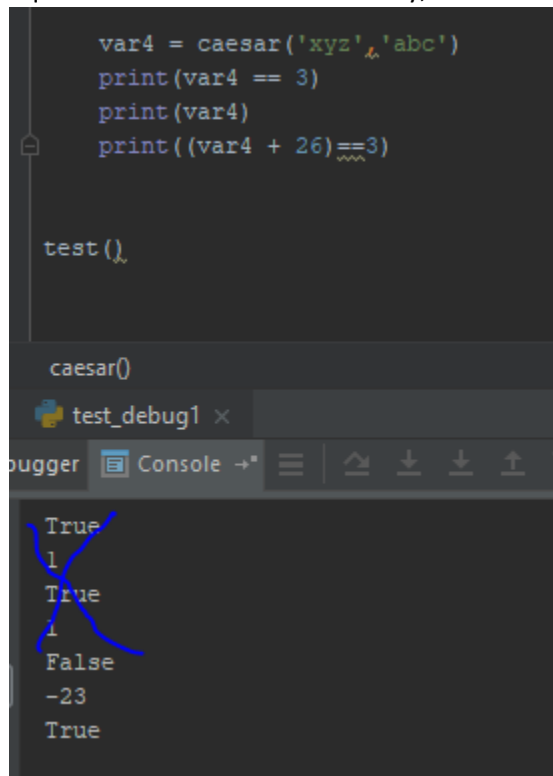
caesar() > for idx in range(len(codeword))

tables

- codeword = {str} 'tca'
- idx = {int} 2
- original = {str} 'ray'
- shift = {int} 2
- temp = {int} -24

As seen above, the variable temp, which is the distance between the coded and original letter is -24 because of a letter had to loop around the alphabet. This bug only happens when letters are coded "past z". To fix this, we must account for a letter's corresponding negative shift. For example. A shift of 2 is

equal to a shift of -24. Additionally, we must account for negative value shifts.



```
var4 = caesar('xyz', 'abc')
print(var4 == 3)
print(var4)
print((var4 + 26) == 3)

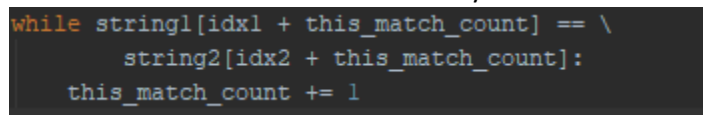
test()

caesar()
test_debug1 x
Debugger Console →
True
1
True
1
False
-23
True
```

As you can see here, when a shift requires a loop in the alphabet in the first character, the function will always return a negative value. To fix this, we must add 26 to all negative outputs. This fulfills the return value requirements ($0 \leq \text{return} \leq 25$)

Problem 2.

One of the first issues I noticed was a syntax error.



```
while string1[idx1 + this_match_count] == \
    string2[idx2 + this_match_count]:
    this_match_count += 1
```

As you can see here, the while statement is separated by a random character and whitespace. This creates a syntax error which can be fixed by merging the statement onto 1 line and deleting the “\” from the line.

Two inputs that work correctly are “god”, “gob” and “colors”, “peanuts”. These two inputs both work because they are the same length. Issues arise when one string is larger than another string. At some point in the comparison, the code references the strings at two identical indexes. If one string is shorter, an error will be produced causing the program to error. To account for this, a check must be included that can account for a possible difference in string length.

```
65 # for all possible string2 start points
66 for idx2 in range(len(string2)-1): idx2: 1
67 # check if these characters match
68 if string1[idx1] == string2[idx2]:
69     this_match_count = 1 this_match_count: 2
70 # see how long the match continues
71 while string1[idx1 + this_match_count] == 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
78 # now return the result
79 return best_length
80
81 def test():
82     print(match('peanut','can'))
83
```

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

Variables

- Special Variables
- best_length = {int} 0
- idx1 = {int} 2
- idx2 = {int} 1
- string1 = {str} 'peanut'
- string2 = {str} 'can'
- this_match_count = {int} 2

As can be seen above, the index of the second string plus the current match count (2) yields an index of 3, which isn't a possible index in a string with a length of 3. To fix this, a check statement should be inserted into the while loop below the line that updates this_match_count. Here the program should check if the next index for both strings. If it does, continue looping, if not, break the loop and continue.