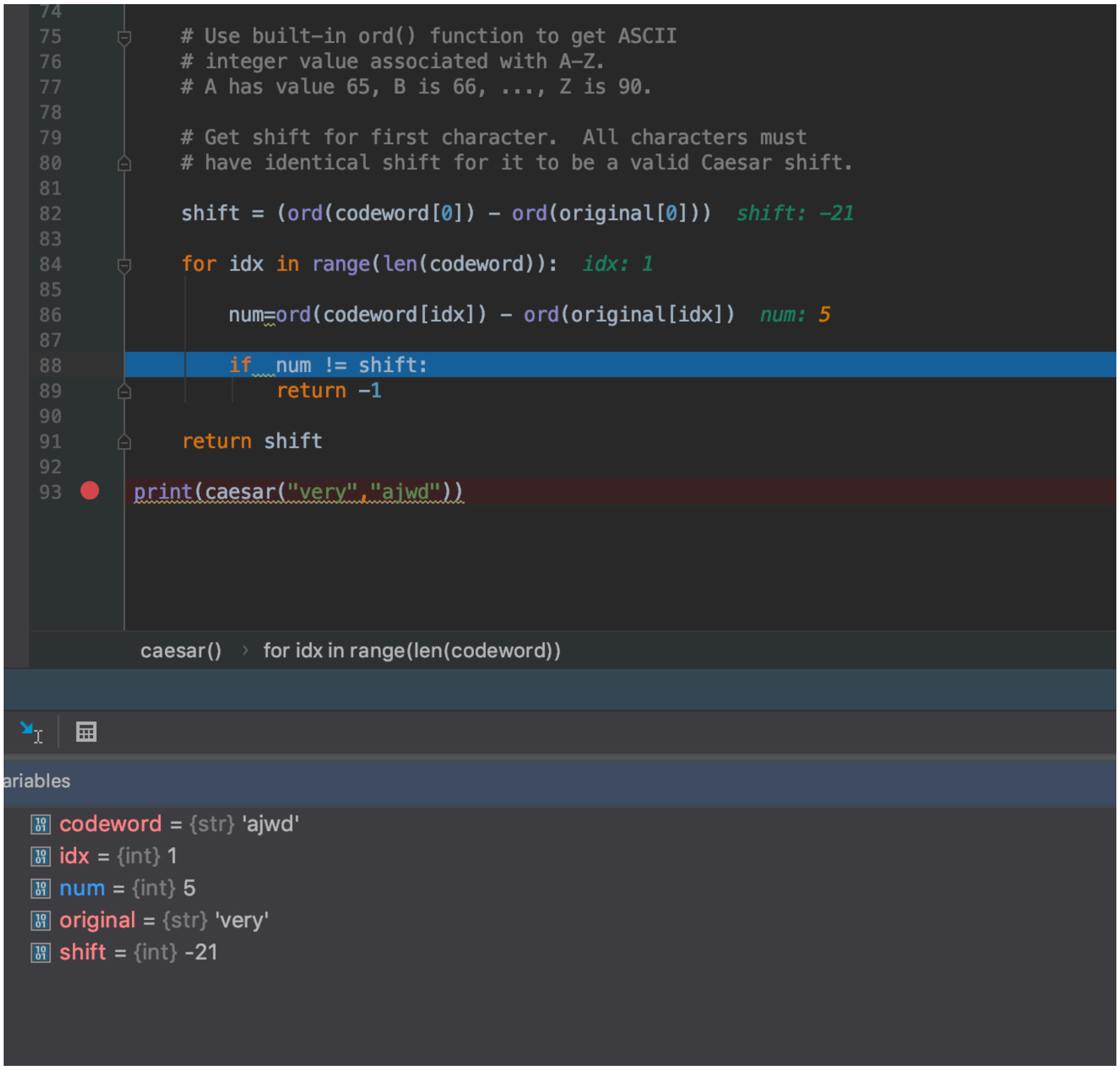
1-
First problem: (hi,mn),(go,lt)
Second problem: (cvhi,cvlk),(goal, gone)

2-
First problem because it does not go the beginning of the alphabets then it will appear 5 not -21.

Second problem because the Consecutive Matching in their word are in the beginning.

3-
The first problem that when the shift variable is negative. You have to add condition when the shift is negative add 26 to it. 26 which is the number of the alphabets
The second problem when the function start count the consecutive matching in some cases one of the strings get out of their range. You have to make break condition before to get out of the range

Problem 1:



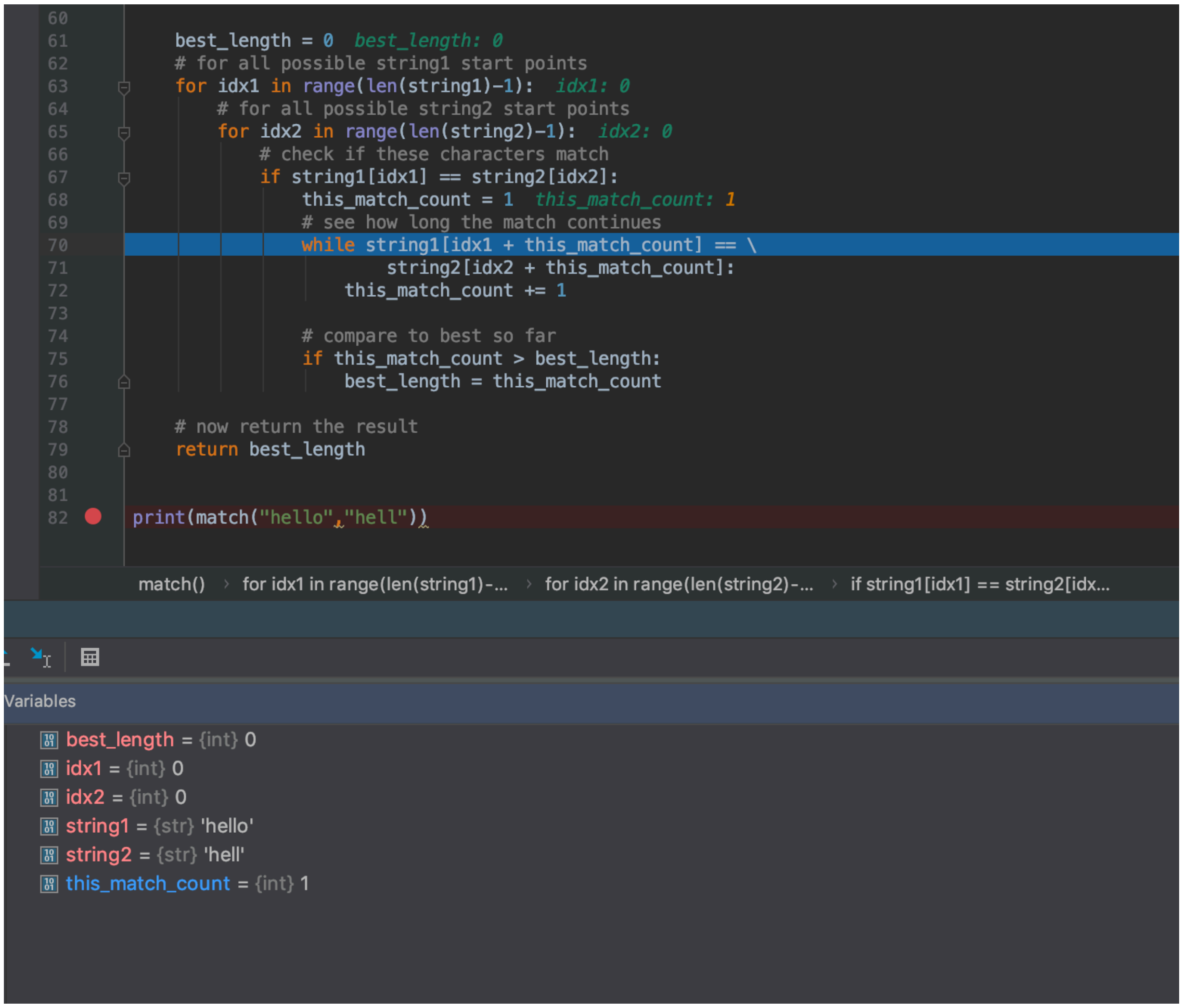
```python
    # Use built-in ord() function to get ASCII
    # integer value associated with A-Z.
    # A has value 65, B is 66, ..., Z is 90.

    # Get shift for first character.  All characters must
    # have identical shift for it to be a valid Caesar shift.

    shift = (ord(codeword[0]) - ord(original[0]))  shift: -21

    for idx in range(len(codeword)):  idx: 1

        num=ord(codeword[idx]) - ord(original[idx])  num: 5

        if  num != shift:
            return -1

    return shift

print(caesar("very","ajwd"))
```

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

ariables

```
codeword = {str} 'ajwd'
idx = {int} 1
num = {int} 5
original = {str} 'very'
shift = {int} -21
```

Problem 2:



```

    best_length = 0  best_length: 0
    # for all possible string1 start points
    for idx1 in range(len(string1)-1):  idx1: 0
        # for all possible string2 start points
        for idx2 in range(len(string2)-1):  idx2: 0
            # check if these characters match
            if string1[idx1] == string2[idx2]:
                this_match_count = 1  this_match_count: 1
                # see how long the match continues
                while string1[idx1 + this_match_count] == \
                        string2[idx2 + this_match_count]:
                    this_match_count += 1

                # compare to best so far
                if this_match_count > best_length:
                    best_length = this_match_count

    # now return the result
    return best_length


print(match("hello","hell"))
```

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

Variables

```
best_length = {int} 0
idx1 = {int} 0
idx2 = {int} 0
string1 = {str} 'hello'
string2 = {str} 'hell'
this_match_count = {int} 1
```