Carson Sklare

2.

A.

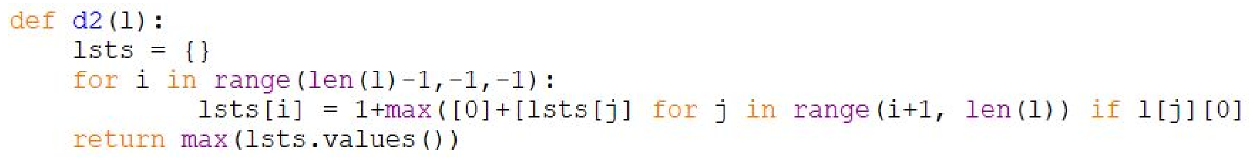
- if you had all the values of D1, can you determine the length of the longest word chain?
    - **You should be able to look at each word to see if they fit with others in order to make the answer on what is the longest chain**
- if you had all the values of D2, can you determine the length of the longest word chain?
    - **You would have to look at the last word to fit in with other values recursively, if programmed dynamically you could check at the end if the value is bigger than what is already recorded to do it much quicker**
- Can the values of D1 be calculated dynamically (i.e. does D1 have an easy recursive definition)?
    - **Seems similar to the naive solution to other problems, could possibly have one**
- Can the values of D2 be calculated dynamically (i.e. does D2 have an easy recursive definition)?
    - **Seems more similar to a recursive problem that can be calculated dynamically**

B.

D2

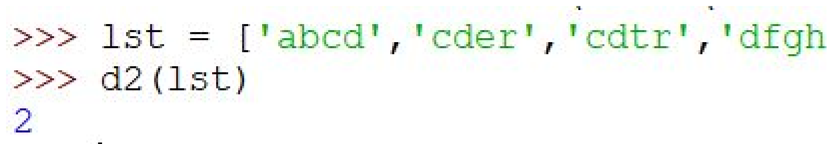
1+ max(d2(i): i[-1:] == i+1[0])

C. code



```
def d2(l):
    lsts = {}
    for i in range(len(l)-1,-1,-1):
            lsts[i] = 1+max([0]+[lsts[j] for j in range(i+1, len(l)) if l[j][0]
    return max(lsts.values())
```

Result:

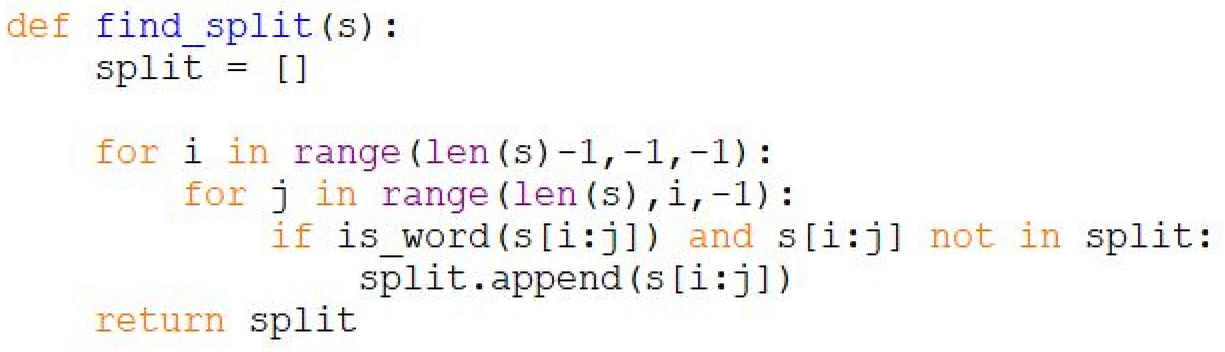


```
>>> lst = ['abcd','cder','cdtr','dfgh
>>> d2(lst)
2
```

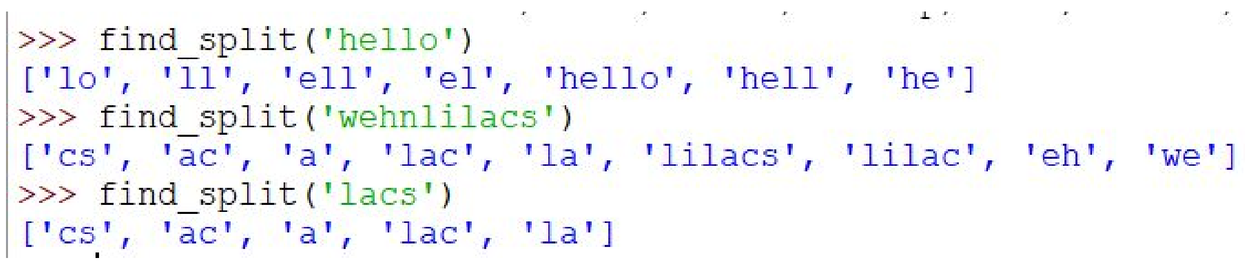
D and E. **Running time will be O(n^2) while space will be n**

3.

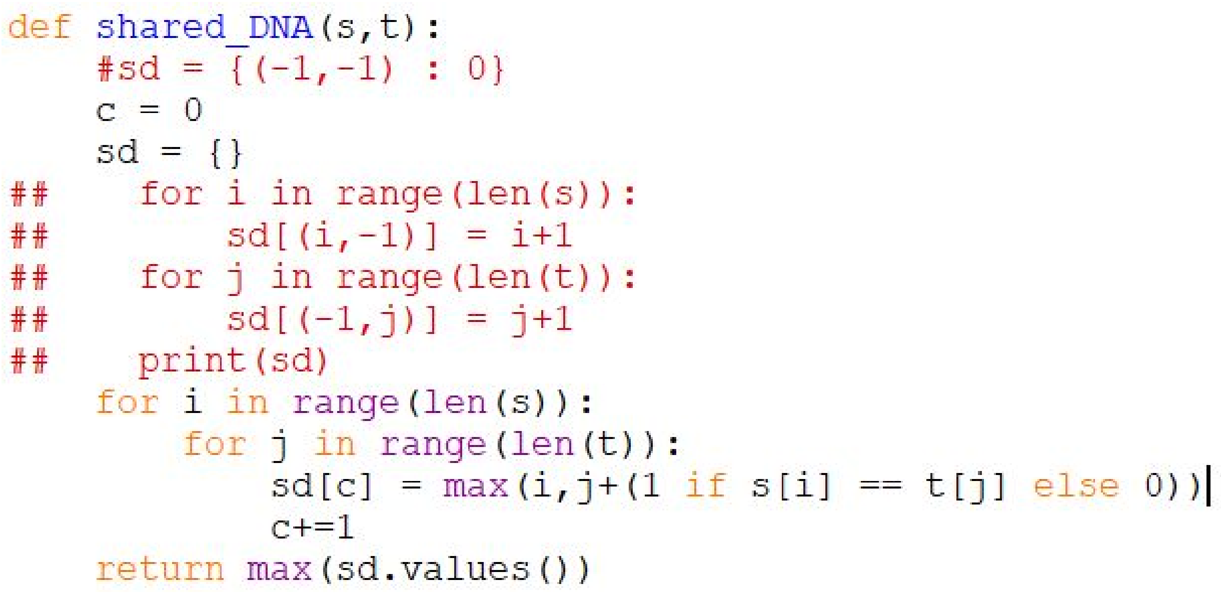
Code:

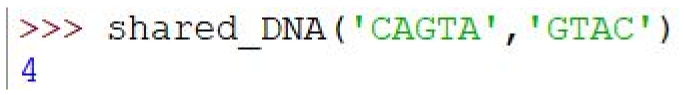

```
def find_split(s):
    split = []

    for i in range(len(s)-1,-1,-1):
        for j in range(len(s),i,-1):
            if is_word(s[i:j]) and s[i:j] not in split:
                split.append(s[i:j])
    return split
```

Results:


```
>>> find_split('hello')
['lo', 'll', 'ell', 'el', 'hello', 'hell', 'he']
>>> find_split('wehnlilacs')
['cs', 'ac', 'a', 'lac', 'la', 'lilacs', 'lilac', 'eh', 'we']
>>> find_split('lacs')
['cs', 'ac', 'a', 'lac', 'la']
```

4.
A and C.**Goal was to go through the words with i and j (would not compile with slicess[:i] and t[:j]), not sure how to do that, code does not seem to work so I left in the stuff that would return correctly. I wanted to have it so it would add the max value of the lengths of correct characters by comparison:**
Code:


```
def shared_DNA(s,t):
    #sd = {(-1,-1) : 0}
    c = 0
    sd = {}
##    for i in range(len(s)):
##        sd[(i,-1)] = i+1
##    for j in range(len(t)):
##        sd[(-1,j)] = j+1
##    print(sd)
    for i in range(len(s)):
        for j in range(len(t)):
            sd[c] = max(i,j+(1 if s[i] == t[j] else 0))
            c+=1
    return max(sd.values())
```
Result:


```
>>> shared_DNA('CAGTA','GTAC')
4
```

B. **Base case would be 0 if s[i] and t[j] are not equal, should recursively add by going through s and s and adding one if the characters are equal. Then finds the max of the values stored through all the traversals**