Levenshtein Distanz

Aufgabe 1:

Schreibe eine Funktion in Pseudocode (oder in Python) die, die Levenshtein-distanz zwischen einem Eingabewort und ein einer Liste die die Wörter eines Englischwörterbuch enthält, berechnen.

```
import requests
online_dictionary = "URL"
response =    requests.get(online_dictionary)
words = response.text.split('\n')

#ToDo make Levenshtein Distance Function
#ToDo create Get Closest Words Function

def GetLevenshteinDistance(word, input_word):
    #ToDo create new 2d Array DONE
    #Define rows, cols DONE
    cols, rows = (len(word)+1, len(input_word)+1)
    matrix = [[0 for _ in range(cols)] for _ in range(rows)]
    #The underscore is a placeholder for the loop variable because we do not need it here
    #For each row it calls this term which returns a list filled of 0's with a length of cols
    #TODO write ascending numbers in the 0 ases DONE
    for i in range(rows):
        matrix[i][0] = i
    for j in range(cols):
        matrix[0][j] = j
    #This represent the cost when created from an empty string
    #Fill the rest of the matrix
    for i in range(1, rows):
        for j in range(1, cols):
```

Merken!

Merken!

Merken

Kernel Logic:

if (word[i] == input_word[j]):
additional_cost = 0
+0 +1
+1 x

else:
additional_cost = 1

defined from Kernelspng from the Mela Classroom

⟶⟶

matrix[i;j] = min(matrix[i-1][j]+1, matrix[i][j-1]+1, matrix[i-1][j-1]+additional_cost)

---

return matrix[-1][-1]

Now lets code the second function

def GetNearestWords(words, input_word, top_n = 10):
# We need a List to save all calculated distances
distances = []

Now we need to iterate over each and every word and calculate the distance
Before we calculate this we strip each entry in the dictionary in order to remove a blank

for word in words:
word.strip()
if word: #returns false/null if the string is empty
distanc = GetLevenshteinDistance(word, input_word)
distances.append(word, distance)

# What is a lamda function?
# A lamda function is a small anonymus (unnamed) function that can take multiple arguments
# but can only have one return expression
distances.sort(key = lamda x: x[1])

Expression to be returned

Merke!

argument

```
return distances[:top_n]
```

returns a sublist with the first n items

```
closest_words = GetNearestWords(words, input_word)
print("the closest 10 words are")
for word, distance in closest words: #For every tuple in the closest_words
    print(f"{word}: {distance}") #Print it with params showing
```