**Python Assignment - 1**

Source Code: [Ritwick's Python Assignment](https://github.com/KumarRitwick/PyhtonAssignment1.git)

Algorithm:

1. For a given string, MEIN DUDE we merge the words => MEINDUDE
2. We then create the weighted array => [0, 1 + x, ...., 20]
3. We pick the first alphabet character and weight and keep is aside
4. We take the remaining characters and weights => EINDUDE and [1 + x, ..., 20]
5. We create the permutations of 2nd and 3rd characters

def getAllAbbIndicesAndWgts(weights):

pairs = []

for i, w1 in enumarate(weights)

for j, w2 in enumarate(weights)

if j<=i or w1 == -1 or w2 == -1:

* 1. continue

pairs.append((i, j, w1 + w2))

return pairs

1. We create the array of all the abs and their scores as such:

```python

def createAllAbs(sentence):

words = sentence.strip().split()

weights = []

for word in words:

weight.append(get\_word\_weights(word))

cmplLetters = "".join(words)

cmplWeight = [ w for weight in weights for w in weight ]

firstCharIdx = getFirstCharIdx(cmplWeight)

rmnLetters = cmplLetters[firstCharIdx + 1:]

rmnWeight = cmplWeight[firstCharIdx + 1:]

allAbsSuffixes = getAllAbbIndicesAndWgts(rmnWeight)

allAbs = []

for each in allAbsSuffixes:

abs = cmplLetters[firstCharIdx] + rmnLetters[each[0]] + rmnLetters[each[1]]

score = each[2]

allAbs.append({ "ABB": abs, "SCORE": score })

allAbs.sort(key=lambda x: x['SCORE'])

return allAbs

```

1. We then add the sentence's abb in the dict: keep them in a sorted list of dicts =>

{

"COOL": [

{ "ABB": "COL", "SCORE" : 5 },

{ "ABB": "COO", "SCORE" : 20 },

...

],

"COLD": [ ... ],

...

}

1. We keep a dict of selected abs: { "COOL": [0], "COLD": [0], ... }
2. We keep another dict to exclude abs: { "COL": 5, ... }

10. We select them as such:

def getSelectedAbs(line, lineAbs, excludeAbs):

if line(lineAbs) <= 0:

return None, excludeAbs

selectedAbsIndices = []

selectedScore = None

for i, abDict in enumarate(lineAbs):

ab = abDict["ABB"]

if ab in excludeAbs:

continue

score = abDict["SCORE"]

if selectedScore is None:

selectedScore = score

if score > selectedScore:

break

execludeAbs[ab] = True

selectedAbsIndices.append(i)

return selectedAbsIndices, execludeAbs

def selectOptimizedAbs(allAbs, allLines):

selectedAbs = {}

toExcludeAbs = {}

for line in allLines:

lineAbs = allAbs[line]

selected, excludes = getSelectedAbs(line, lineAbs, toExcludeAbs)

toExcludeAbs = excludes

selectedAbs[line] = selected

return selectedAbs

1. We finally get the abs of each line

selectedAbs = {}

for line, scoreIndices in allOptimisedAbbs.items():

if scoreIndices is None:

selectedAbs[line] = None:

else:

selectedAbs[line] = [ all\_abs[line][scrIdx] for scrIdx in scoreIndices]

print(selectedAbs)