## Task 1: [1 mark]

The code provided below is for a **recursive function** that computes the edit distance between two strings *x* and *y*. Modify this code such that:

(a) the function measures a **weighted edit distance**, where

insertion cost = 1, deletion cost = 1

substitution cost = 1, if a letter is substituted with another letter

substitution cost = 2, if a letter is substituted with a number

(b) use the [time.time() method](https://stackoverflow.com/questions/7370801/how-to-measure-elapsed-time-in-python) to measure the execution time.

**Note1**: in case you are not familiar with recursive functions, you must first watch the short optional video in the lesson which reviews this topic. Once you understand how to compute ***n*!** using a recursive function, you can solve this task quickly.

**Note2 (optional)**: More advanced coders can consider adding [tracing](https://www.codementor.io/@dmitrybelaventsev/python-trace-recursive-function-tkq79m4so) capabilities to their recursive function. This optional task carries no marks but could be an insightful way to understand recursion.

import time

#How many insertions, deletions, or substitutions does it take to turn x into y?

def edDistRecursive(x, y):

# If either x or y are empty, then one or more (depending on their length) INSERTIONS or DELECTIONS are needed to transform x to y.

if len(x) == 0:

#print(f'\t Converting "{x}" TO "{y}" requires {len(y)} INSERTIONS, therefor it costs {len(y)}')

return len(y)

if len(y) == 0:

#print(f'\t Converting "{x}" TO "{y}" requires {len(x)} DELETIONS, therefor it costs {len(x)}')

return len(x)

# When neither x or y are empty, we must compute the cost of each operation and find the least costly one.

delta = 1 if x[-1] != y[-1] else 0

diagonal\_or\_substitution\_cost = edDistRecursive(x[:-1], y[:-1]) + delta #what's the cost of SUBSTITUTING the last character of x with the last character of y

vertical\_or\_deletion\_cost = edDistRecursive(x[:-1], y) + 1 #what's the cost of DELETING the last character of x

horizontal\_or\_insertion\_cost = edDistRecursive(x, y[:-1]) + 1 #what's the cost of INSERTING the last character of x into y

# what's the least costly operation?

minValue = min(diagonal\_or\_substitution\_cost, vertical\_or\_deletion\_cost, horizontal\_or\_insertion\_cost)

return minValue

Expected Output:

Edit Distance between "intention" and "execution": 5

--- Executed in: 1.1639270782470703 seconds ---

## Task 2: [4 marks]

To complete this task you need to install two libraries:

* Install the First and Last Names Dataset (<https://github.com/philipperemy/name-dataset>). (!pip install names-dataset)
* Install the jellyfish library for doing approximate and phonetic matching of strings (<https://github.com/jamesturk/jellyfish>) (!pip install jellyfish)

1. Write a function that takes your firstName and lastName as input and checks if they exist in the First and Last Names dataset.
2. Write a function that takes your firstName and lastName as input and prints out their Soundex codes (use the Soundex function in the jellyfish library).
3. Write a function that takes your firstName as input and prints out a list of first names in the dataset whose Soundex codes are the same as your firstName’s Soundex code.
4. Write a function that takes your firstName and lastName as input and returns their Levenshtein Distance (use the Levenshtein Distance function in the jellyfish library).

| Task 3: [5 marks] The code provided [**HERE**](https://gist.github.com/ajoorabchi/271f246e8bd70adfd8bf2b2b4066234b) is for a function that computes the edit distance between two strings ***x*** and ***y***,using dynamic programming. Modify this code such that:  (a) the function measures the **Levenshtein distance**, i.e. insertion cost = 1, deletion cost = 1, substitution cost = 2  (b) use [%time](https://ipython.readthedocs.io/en/stable/interactive/magics.html?highlight=%25time#magic-time) or [time.time() method](https://stackoverflow.com/questions/7370801/how-to-measure-elapsed-time-in-python) to measure the execution time.  (c) Augment the code such that it outputs a DP table including backtrace pointers in each cell (as shown in the sample output below).  **Note1**: Please see [Chapter 2, section 2.5.1](https://web.stanford.edu/~jurafsky/slp3/2.pdf) for more details on The Minimum Edit Distance Algorithm and backtrace.  **Note2:** the direction of vertical arrows (up/down) depends on the orientation of the DP table (e.g. see the two sample tables on the right), therefore it could be easier if you think of possible operations as:   1. **Substitution OR no change (diagonal):** Cost ( ↖ substitution ) = 2 **OR** Cost ( ↖ no-change ) = 0 2. **Deletion (vertical: down/up):** Cost( ↑↓ deletion ) = 1 3. **Insertion (horizontal):** Cost( ← insertion ) = 1   Expected Output:  ---------------------------------------------------------------------------------------------------------------  Completed DP table after all the subproblems are solved  0↑ 1← 2← 3← 4← 5← 6← 7← 8← 9←  1↑ 2←↖↑ 3←↖↑ 4←↖↑ 5←↖↑ 6←↖↑ 7←↖↑ 6↖ 7← 8←  2↑ 3←↖↑ 4←↖↑ 5←↖↑ 6←↖↑ 7←↖↑ 8←↖↑ 7↑ 8←↖↑ 7↖  3↑ 4←↖↑ 5←↖↑ 6←↖↑ 7←↖↑ 8←↖↑ 7↖ 8←↑ 9←↖↑ 8↑  4↑ 3↖ 4← 5←↖ 6← 7← 8←↑ 9←↖↑ 10←↖↑ 9↑  5↑ 4↑ 5←↖↑ 6←↖↑ 7←↖↑ 8←↖↑ 9←↖↑ 10←↖↑ 11←↖↑ 10↖↑  6↑ 5↑ 6←↖↑ 7←↖↑ 8←↖↑ 9←↖↑ 8↖ 9← 10← 11←↑  7↑ 6↑ 7←↖↑ 8←↖↑ 9←↖↑ 10←↖↑ 9↑ 8↖ 9← 10←  8↑ 7↑ 8←↖↑ 9←↖↑ 10←↖↑ 11←↖↑ 10↑ 9↑ 8↖ 9←  9↑ 8↑ 9←↖↑ 10←↖↑ 11←↖↑ 12←↖↑ 11↑ 10↑ 9↑ 8↖  ---------------------------------------------------------------------------------------------------------------  Levenshtein distance between "intention" and "execution": 8  --- Executed in: 0.015343189239501953 seconds --- | **Source = intention, Target = execution** |
| --- | --- |

## </END OF ETIVITY>

Resources for Data Science and Statistical Learning students:

* [A gentle introduction to OOP in Python (classes, methods, attributes, objects)](https://www.youtube.com/watch?v=f0TrMH9s-VE)

Teaching team:

* Level of difficulty (easy/fair/hard)
* Task4 ([Weighted Minimum Edit Distance](https://youtu.be/gx7Fkf1n_5w) video needs to be added to the main material): Augment your code from Task1 or task2 such that it computes a weighted edit distance, where
  1. the cost of substituting alphabetic characters [A-Za-z] is 2; and
  2. the cost of substituting digits [0-9] is 3