Name:
ID:

M5 Final Exam - Free Response

Free Response Questions

1. Create a function named ddx that finds the derivative of a polynomial:

$$\frac{d}{dx}(a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} + a_nx^n) = a_1 + (2)a_2x + \ldots + (n-1)a_{n-1}x^{n-2} + (n)a_nx^{n-1}$$

- We will consider polynomials built only from monomials that have natural number powers.
- The function should take one parameter P which is a list of floats.
- The first element of P corresponds to the a_0, the second corresponds to a_1, …, the last element corresponds to a_n. P is a list of the polynomial coefficients.
- The function should return a list which represents the coefficients of the derivative polynomial.

Code

Test Case	P	Derivative
Test 1	[2,3,13.5]	
Test 2	[-0.5, 3/2, 13, 100, 15]	

Solution:

```
# Nick Sebasco
#
def ddx(P:list)->list:
    '''Compute the derivative of a polynomial.
    '''
    return [i * P[i] for i in range(1,len(P))]


P = [2, 3, -1] # -x^2 + 3x + 2
dPdx = ddx(P)  # -2x + 3
print(dPdx)
```

2. The following variable **alpha** contains a slice of the covid19 alpha variant's RNA base pair sequence. Let **mutant_00** and **mutant_01** represent mutations to the alpha variant. Compute the Levenshtein distance and determine which mutation is most similar to **alpha**.

Covid19 variant	**First ~120 base pairs**
alpha	**"attaaaggtt tataccttcc caggtaacaa accaaccaac tttcgatctc ttgtagatct gttctctaaa cgaactttaa aatctgtgtg gctgtcactc ggctgcatgc ttagtgcact"**
mutant_00	**"attaaaggtt tactacacaacc cacgtaacaa accaaccaac tttcgatctc ttgtagatct gttggtctaaa cgaactttaa aactctgtgtg gctgtcactc ggctgcatgc tagtgcact"**
mutant_01	**"attataggtt tatagcttcc cagtaccacaa accacccaac tttcgatctc tggtctatct tctaaggaaa cgaaactttaa aatctgtgtg cctgtcactc ggctgcatgc tgcgtgcact"**

Code

Answer

Solution:

```
# Nick Sebasco
#
import numpy as np
def lev(s1: str, s2: str) -> np.matrix:

    '''AKA: Edit distance

    levenshtein distance between two sequences s1, s2 is the minimum number of single

    element (insert/ delete/ substitution) required to transform s1 -> s2.

    '''

    # 1. initialize matrix

    T = np.zeros((len(s1) + 1, len(s2) + 1))

    for i in range(max(len(s1)+1, len(s2)+1)):

        if i < len(s1)+1:

            T[i, 0] = i

        if i < len(s2)+1:

            T[0, i] = i

    # 2.

    for i in range(len(s1)):

        for j in range(len(s2)):

            T[i + 1, j + 1] = min(T[i, j + 1], T[i + 1, j], T[i, j]) + 1 if s1[i] !=
s2[j] else T[i, j]

    return T


# solution to problem:

a = 'attaaaggtt tataccttcc caggtaacaa accaaccaac tttcgatctc ttgtagatct gttctctaaa
cgaactttaa aatctgtgtg gctgtcactc ggctgcatgc ttagtgcact'

m1 = 'attaaaggtt tactacacaacc cacgtaacaa accaaccaac tttcgatctc ttgtagatct gttggtctaaa
cgaactttaa aactctgtgtg gctgtcactc ggctgcatgc tagtgcact'

m2 = 'attataggtt tatagcttcc cagtaccacaa accacccaac tttcgatctc tggtctatct tctaaggaaa
cgaaactttaa aatctgtgtg cctgtcactc ggctgcatgc tgcgtgcact'


d1 = lev(a, m1)

d2 = lev(a, m2)

print(d1)

print()

print(d2)

# So how do you tell which mutant is more similar?  We need the smallest distance.
```

```
# The final value in the matrix is the distance.
```