In [2]:
```
import numpy as np
from sklearn.preprocessing import normalize
```

In [3]:
```
# Constants used in this exercise
# Fill in all of the ...s/TODOs
length = 101
width = 8
```

In [4]:
```
# Read in the files using BioPython
# TODO:
```

In [5]:
```
# Extract strings of sequences from the above files
# TODO:

seqs = ...
```

In [6]:
```
# Initialize p with a uniform background

# Helper dict for indexing
let = {"A":0, "C":1, "G":2, "T":3}

def init_p(seqs, l, w):
    p = np.zeroes((4, w+1))

    # set a uniform background
    for i in let.keys():
        # TODO:
        p[let[i]][0] = ...
        #p[let[i]][0] = 0.25 #ANSWER

    # set motif positions
    for i in range(l-w):
        for sequence in seqs:
            for j in range(w):
                p[ let[ sequence[i + j] ] ][j + 1] += 1

    # normalize columns to sum to 1
    p = normalize(p, axis = 0, norm = 'l1')

    return p
```

In [18]:
```
# Define a general function to run EM

def run_EM(l, w, seqs, init_p, up_prob, up_motif, epsilon = 0.005):
    no_change = False

    # set p_t_1 to the initial p_0 using init_p
    # TODO:
    p_t_1 = ...

    #p_t_1 = init_p(l, w, seqs) #ANSWER

    while not no_change:

        # Label the following steps as E step or M step in the comment preceding

        # TODO:
        #

        # E step #ANSWER
        z_t = up_motif(l, w, p_t_1, seqs)

        # TODO:
        #

        # M step #ANSWER
        p_t = up_prob(w, z_t, seqs)

        diff = np.subtract(p_t, p_t_1)

        # Carry out the next steps to determine if every element in diff is < ep
silon
        # TODO:
        if ...:

        #if 4*(w+1) == np.sum(diff<epsilon) #ANSWER
            no_change = True
        else:
            #update p_t_1 so we can keep iterating
            #TODO:
            ...

            #p_t_1 = p_t #ANSWER

    return p_t, z_t
```

In [13]:
```
# Define a function to update z

def up_motif(l, w, p_t_1, seqs):
    z_t = np.zeros((len(seqs), l-w))

    for i, sequence in enumerate(seqs):
        for j in range(l-w):

            # Fill in z_t using p_t_1
            # Ignore background as we're assuming 0.25 for all 4
            # TODO:
            z_t[i][j] = ...

            #z_t[i][j] = np.multiply([p_t_1[let[letter]][position] for position,
letter in enumerate(sequence[j:j+w])]) #ANSWER

            ### 2ND OPTION ###
            #Expanded

            # Splice sequence to get the w letters starting at index j
            # TODO:
            motif = ...

            #motif = sequence[j:j+w] #ANSWER

            mul = 1

            for position, letter in enumerate(motif):

                # Index into p_t_1 using the position and letter
                # TODO:
                mul *= p_t_1[...][...]

                #mul*= p_t_1[let[letter]][position] #ANSWER

            # Set z_t
            z_t[i][j] = mul



    # Normalize z_t so each row sums to 1
    # Hint: Look at init_p's last line, and axis = 1 => apply to rows
    # TODO:
    z_t = ...

    #z_t = normalize(z_t, axis = 1, norm = 'l1') #ANSWER

    return z_t
```

In [ ]:
```
# Define a function to update p

def up_prob(w, z_t, seqs):
    p_t = np.zeros((4, w+1))

    n = np.zeros((4, w+1))

    # Fill in n for k > 0
    for k in range(1, w+1):
        for letter in let.values():
            sum_z = 0
            for i, sequence in enumerate(seqs):

                # Write j_vals to obtain a list of all indices
                # of where letter appears in sequence
                # TODO:
                j_vals = ...

                #j_vals = [i for i in range(len(sequence)) if sequence[i] == let
ter] #ANSWER

                for j in j_vals:

                    # Add the element form z_t to sum_z
                    # TODO:
                    sum_z += ...

                    #sum_z += z_t[i][j] #ANSWER

            # Fill in the correct indices
            # Remember to switch back from letter to a number
            # TODO:
            n[...][...] = sum_z

            #n[let[letter]][k] = sum_z #ANSWER

    # Fill in n for k == 0

    # Helper variable to make the next step easier
    joined_seq = "".join(seqs)

    # Import a method to count all occurances of A,C,G,T in all seqs
    # totals should be a dict with letter mapping to a count
    # (the proper method will do it for you)
    # TODO:
    counts = ...

    #counts = Counter(joined_seq) #ANSWER

    # Sum across the rows of n as filled in so far
    # TODO:
    sum_n_j = ...

    #sum_n_j = np.sum(n, axis = 1) #ANSWER


    for letter in let.values():

        #

        # Fill in the correct indices
        # Remember to switch back from letter to a number
        # TODO:
```