```python
# -------------------------------------
# Created by Arjun Batra 08/03/2022
# Final Edits by Arjun Batra 08/10/2022
# Harvard iGEM 2022
# -------------------------------------

from Bio.Seq import Seq
from Bio import SeqIO
from Bio import Align
from Bio.Align import substitution_matrices
import numpy as np
import math
import matplotlib.pyplot as plt


def save_alignment(alignment_list, file, title):
    seq_size = len(alignment_list[0])

    file.write("=========================Alignment=========================\n")
    file.write(title + "\n")
    for i in range(0, seq_size, 50):
        start_res = i+1
        end_res = i+50
        if end_res > seq_size:
            end_res = seq_size
        for seq in alignment_list:
            file.write(str(start_res) + " " + seq[start_res-1:end_res] + " " +
             str(end_res) + "\n")
        file.write("\n")
    file.write("===========================================================\n")
    file.write("\n")
    file.write("\n")

# Find residue number of an amino acid (classified by its index in the aligned
 sequence) in an an aligned sequence that contains gaps
def find_resn(aligned_seq, align_index):
    resn = 1
    for i in range(len(aligned_seq)):
        if (i == align_index):
            return resn
        if (aligned_seq[i] != "-"):
            resn += 1

# Gets the residue's number for resiudes in the format like "Y27"
def get_resn(residue):
    return int(residue[1:])

# Gets the residue's amino acid for residues in the format like "Y27"
def get_aa(residue):
    return residue[0]
```

```
# Determining frequency of amino acids from a list of resiudes (in format like
 "Y27"), ignoring gaps "-" in the list
def aa_freq(res_list):
    freq = {}
    for res in res_list:
        if (res != "-"):
            aa = get_aa(res)
            if aa in freq.keys():
                freq[aa] += 1
            else:
                freq[aa] = 1
    return freq

# Determining Shannon entropy based on a list of frequencies
def sh_entropy(freq_list):
    tot_n = sum(freq_list)
    # Determine the Shannon entropy
    shannon = 0
    for freq in freq_list:
        prob = freq/tot_n
        shannon = shannon - (prob*math.log(prob))
    return shannon



# Determining initial pairwise alignment matrix

# Files
ref_fasta = "ste2_ref.fasta" # Has one nucleotide sequence that is being
 engineered
evo_fasta = "ste2_evo.fasta" # Has mulitple nucleotide sequences that are
 evolutionarily related to ref_fasta

# Setting up pairwise alignment parameters
aligner = Align.PairwiseAligner()
aligner.mode = "global"
aligner.open_gap_score = -5
aligner.extend_gap_score = -2
aligner.substitution_matrix = substitution_matrices.load("BLOSUM62")


# Reads the reference sequence from its fasta file
ref = SeqIO.read(ref_fasta, "fasta")

# Initializes pairwise alignment matrix with ref as first (reference) entry
ref_seq_str = str(ref.seq)
pa_mat = [[ref_seq_str[i] + str(i+1)] for i in range(len(ref_seq_str))]

# Goes through each of the evo sequences and performs alignment with ref
# As it performs alignment, does the following:
# - Writes to pairwise alignment file pair_align.txt
# - Adds to the pairwise alignment matrix the residues that align with the
 one's in ref (ignores residues in evo that align with gaps in ref)

pa_file = open("pair_align.txt","w")

for evo in SeqIO.parse(evo_fasta, "fasta"):
    evo_seq_str = str(evo.seq)
    score = aligner.score(ref_seq_str, evo_seq_str)
    align = aligner.align(ref_seq_str, evo_seq_str)

    align_list = str(align[0]).split()
    save_alignment(align_list, pa_file, "Score: " + str(score))
    ref_align = align_list[0]
    evo_align = align_list[2]


    for i in range(len(ref_align)):
        if (ref_align[i] != "-"):
            ref_seq_index = find_resn(ref_align, i) - 1
            if (evo_align[i] != "-"):
                evo_seq_aa = evo_align[i]
                evo_seq_resn = find_resn(evo_align, i)
                pa_mat[ref_seq_index].append(evo_seq_aa + str(evo_seq_resn))
            else:
                pa_mat[ref_seq_index].append("-")

'''
for i in range(len(ref_seq_str)):
    print(pa_mat[i])
'''

# Determine residues involved in binding (both to alpha factor and cystatin),
 both directly in ref and evolutionarily on ref
bind_dict = {
"ref": ["Q51", "Y98", "Y101", "Y106", "Y111", "Y128", "T131", "N132", "Q135",
 "T192", "V196", "S197", "A198", "T199", "Q200", "D201", "F204", "N205",
 "P270", "N271", "D275"],
1: ["M88", "N186", "Y91", "D31", "S185", "T125", "I181", "E262", "V122",
 "Q105", "Q194", "Y107", "N177", "V39", "Y35", "A184", "F96", "L173", "N264",
 "R170", "Y102", "S118", "A180", "F92", "Y43", "I191", "F187"],
10: ["Q134", "N205", "T198", "H93", "F97", "T183", "N111", "K47", "T130",
 "R201", "Y127", "F54", "D275", "Q50", "S199", "N131", "E39", "F112", "Y100",
 "F43", "K46", "S187", "F204"],
11: ["E50", "M112", "Q134", "Y110", "I28", "T183", "G114", "N111", "K47",
 "D274", "T33", "Y29", "S27", "Y127", "F54", "Y199", "F203", "Y97", "Y100",
 "S187"],
12: ["Q134", "N205", "T198", "H93", "F97", "T183", "N111", "K47", "T130",
 "R201", "Y127", "I42", "D275", "Q50", "S199", "E39", "N131", "F112", "Y100",
 "F43", "K46", "S187", "F204"],
13: ["M88", "N186", "Y91", "D31", "S185", "I181", "K260", "E262", "V122",
 "Q105", "Q194", "P259", "R38", "Y107", "N177", "V39", "Y35", "A184", "F96",
 "L173", "Y102", "S118", "A180", "F92", "Y43", "F32", "F187", "N97", "I191",
 "A188"],
14: ["G184", "L88", "I178", "M101", "Y114", "T188", "D187", "T263", "N98",
 "S174", "R34", "Y92", "S29", "N185", "H80", "T33", "T177", "F41", "S183",
 "P186", "Y84", "Y87", "S266", "D262", "R44", "Q37", "L30", "N118"],
15: ["M88", "N186", "Y91", "S185", "E31", "F102", "D28", "I181", "E262",
 "Q194", "P259", "Y107", "N177", "V39", "I122", "Y35", "L173", "R170", "S118",
 "F92", "Y43", "F32", "F187", "Y261", "R188"],
16: ["V259", "W260", "T152", "E234", "N270", "I266", "P256", "H137", "S262",
 "N140", "N230", "D253", "V151", "Y148", "V255", "I258", "L142", "A150",
 "R251"],
17: ["Q200", "A198", "Y106", "L113", "A112", "P270", "Y98", "D201", "Y111",
 "Y30", "S47", "N132", "Y128", "N271", "D275", "S197", "E40", "Q51", "T131",
 "L44", "Y101", "F55", "T199", "I36", "F204"],
18: ["Q200", "A198", "Y106", "P270", "Y111", "D201", "Y98", "S47", "N132",
 "G43", "Y128", "N271", "D275", "S197", "Q51", "T131", "L44", "Y101", "F55",
 "T199", "F204"]
}

ref_list_res = list(np.array(pa_mat)[:, 0])
ref_list_bind_res = []
ref_list_bind_loc = [" "] * len(ref_list_res)
num_seq = len(np.array(pa_mat)[0, :])

for i in range(num_seq):
    # Finds the binding residues belonging to the aligned sequence (ref or evo)
    aligned_list_bind_res = bind_dict[list(bind_dict.keys())[i]]
    # Gets the list of residues and gaps of the aligned sequence that matches
     with ref (does not include residues that were aligned with a gap in ref)
    aligned_list_match_res = list(np.array(pa_mat)[:, i])

    # Goes through each binding residue in the aligned sequence
    for res in aligned_list_bind_res:
        # Checks if that binding residue does not align with to a gap in ref
         (in aligned_list_match_res)
        if (res in aligned_list_match_res):
            # Gets the index where the residue matches with ref
            # print("Complex " + str(list(bind_dict.keys())[i]) + ": " +
             str(res))
            match_index = aligned_list_match_res.index(res)

            # Finds the corresponding ref residue and adds it non-repetitively
             to the list of ref residues that are classified as "binding"
            ref_bind_res = ref_list_res[match_index]
            # print("Matching Ref Residue: " + str(ref_bind_res))
            if (ref_bind_res not in ref_list_bind_res):
                ref_list_bind_res.append(ref_bind_res)
                ref_list_bind_loc[match_index] = "^"
```

```python
ref_list_bind_res.sort(key=get_resn)
# print(ref_list_bind_res)

mut_loc_file = open("mut_loc.txt", "w")
save_alignment([ref_seq_str, "".join(ref_list_bind_loc)], mut_loc_file,
 "Location of regions for mutations (indicated by ^)")

# Find and graph Shannon entropy of those residues
# Find candidate amino acid for each of the binding positions
ref_list_bind_sh_entropy = []
ref_list_bind_candidate = []
# print(ref_list_res)
for bind_res in ref_list_bind_res:
    bind_index = ref_list_res.index(bind_res)
    amino_acid_frequency = aa_freq(pa_mat[bind_index])
    ref_list_bind_sh_entropy.append(sh_entropy(list(amino_acid_frequency
     .values())))
    ref_list_bind_candidate.append(list(amino_acid_frequency.keys()))

# print(ref_list_bind_sh_entropy)
for i in range(len(ref_list_bind_candidate)):
    print(ref_list_bind_res[i] + ": " + str(ref_list_bind_sh_entropy[i]))

# plt.bar(ref_list_bind_res, ref_list_bind_sh_entropy)
# plt.show()

# Determine candidate amino acids for


# Determine mutation matrix
amino_acids = ["A", "R", "N", "D", "C", "Q", "E", "G", "H", "I", "L", "K",
 "M", "F", "P", "S", "T", "W", "Y", "V"]
mut_matrix = np.zeros((len(ref_list_res), 20))
sh_cutoff = 0.4

# File to record the new possible amino acids for each position
mut_new_aa = open("mut_new_aa.csv", "w")
mut_new_aa.write("Residues\n")

pos_for_mut = ""

for res_index in range(len(ref_list_res)):
    residue = ref_list_res[res_index]
    candidate = [get_aa(residue)]
    if (residue in ref_list_bind_res):
        bind_index = ref_list_bind_res.index(residue)
        sh_e = ref_list_bind_sh_entropy[bind_index]
        if (sh_e >= sh_cutoff):
            candidate = ref_list_bind_candidate[bind_index]
            pos_for_mut += str(get_resn(residue)) + "+"
    num_candidate = len(candidate)
    eq_prob = 1/num_candidate
    for aa in candidate:
        aa_index = amino_acids.index(aa)
        mut_matrix[res_index, aa_index] = eq_prob

    # Creates list of candidates that do not include the original
    new_mutations = candidate[:]
    new_mutations.remove(get_aa(residue))
    if (len(new_mutations) == 0):
        mut_new_aa.write(str(res_index+1) + "\n")
    else:
        mut_new_aa.write(str(res_index+1) + "," + ",".join(new_mutations) +
         "\n")

print(pos_for_mut)
# print(mut_matrix)


mut_file = open("mut_matrix.csv", "w")
mut_file.write("Residue," + ",".join(amino_acids) + "\n")
for i in range(len(mut_matrix)):
    mut_file.write(str(i+1) + "," + ",".join([str(prob) for prob in
     mut_matrix[i, :]]) + "\n")

'''
fig, ax = plt.subplots()
im = ax.imshow(mut_matrix)
ax.set_xticks(np.arange(20), labels=amino_acids)
ax.set_yticks(np.arange(len(ref_list_res)), labels=range(1, len(ref_seq_str) +
 1))
ax.figure.colorbar(im)
plt.show()
'''
```