1 Exercise 1: Retrieving information for the protein TMM25_HUMAN

This code below does the following:

1. Get the sequence (`getSequence`)
2. Split by lines (`stream.splitlines()`)
3. Trim all carriage returns (`process`)
4. Get the id (`getid`)
5. Get signal (`get_signal_range`)
6. Get sequence (`get_aa_seq`)

The output is:

```
Currently processing the sequence with the name UNIPROT:TMM25_HUMAN
There is a signal peptide ending at  position 26
The signal peptide looks like this: MALPPGPAALRHTLLLLPALLSSGWG
```

The code:

```python
def exercise_1(query_seq):
    def process(txt):
        """make it a bit cleaner"""
        data = []
        for line in txt:
            data.append([s.strip() for s in line.split(" ") if s])
        return data

    def getid(lines):
        for line in lines:
            if line[0] == "ID":
                return line[1]

    def get_aa_seq(lines, signal_start, signal_end):
        aa_start = 0
        aa_end = 0
        for lineno, line in enumerate(lines):
            if line[0] == "SQ":
                # everything from SQ to end is protein sequence
                aa_start = lineno
            if line[0] == "//":
                aa_end = lineno

        aa = "".join(
            [item for sublist in lines[aa_start + 1 : aa_end] for item in sublist]
        )
```

```
        # minus 1, lists start at 0
        return aa[signal_start - 1 : signal_end]

    def get_signal_range(lines: list[list[str]]) -> tuple[int, int]:
        for line in lines:
            if line[0] == "FT" and line[1] == "SIGNAL":
                r = line[2]
                [s, e] = r.split("..")
                return int(s), int(e)

        raise RuntimeError("Did not find signal range")

    stream = getSequence(query_seq, format="txt")
    txt = stream.splitlines()
    lines = process(txt)
    seqid = getid(lines)
    sig_start, sig_end = get_signal_range(lines)
    aa_seq = get_aa_seq(lines, sig_start, sig_end)

    print(f"Currently processing the sequence with the name UNIPROT:{seqid}")
    print(f"There is a signal peptide ending at  position {sig_end}")
    print(f"The signal peptide looks like this: {aa_seq}")
```

2 Exercise 2: Retrieve and process information about multiple members of the same protein family

Here is the commented code to fetch the D amino acid oxidases that have been reviewed and the species they occur in.

After running the code, it found 11 exemplars in E. coli.

```
def exercise_2():
    """ Usage:
    exercise_2()
    """
    # list of `Sequence` class with useful information and methods about a given sequence
    seqs = []
    # list of ids representing an organism that contains DAO and has been reviewed
    names = searchSequences("family:DAO AND reviewed:true")
    #  count of occurrences of E. Coli DOA protein
    cnt = 0
    # get each sequence data from uniprot
    for name in names:
        seq = getSequence(name)
        # add to list for writing to file later
        seqs.append(seq)

        # ensure start of sequence is found in data
```

```
        start_index = seq.annot.find("OS=")

        # it's found!
        if start_index != -1:
            end_index = seq.annot.find("=", start_index + 3)

            # ensure end of sequence is found in data
            if end_index == -1:
                end_index = len(seq.annot)
            # python slicing to get annotation data
            species = seq.annot[start_index + 3 : end_index]
            # print name of seq
            print(seq.name, "\t", species)

            # if it's E. Coli, increment  count
            if species.startswith("Escherichia coli"):
                cnt = cnt + 1
        else:
            print(seq.name, "\tno species")

    # write to file
    writeFastaFile("dao.fa", seqs)

    if cnt > 0:
        print("Found %d exemplars in E. coli" % cnt)
```

3 Exercise 3: Processing the FASTA format

The fasta file format contains genome sequence data. Format is a (>) followed by a description of the sequence (on the same line). The following line is the sequence. Multiple sequences can be included in a single file. Empty lines are ignored.

The example provided will fail because the second sequence has leading white space before the (>) character.

4 Exercise 4: Find two putative homologs to *your* D-amino-acid oxidase

4.1 Part A

1. My sequence number: 83. Sequence: sp|B1JGH2|MNMC_YERPY
2. Sequence 1: sp|A4TM73|MNMC_YERPP. Percent: 1.0
3. Sequence 2: sp|A9R7V8|MNMC_YERPG. Percent: 1.0

4.2 Part B

Here is my code.

```
def exercise_4(my_sequence_number, seqs, sub_matrix):
    """
    find two putative (assumed to be related) homologous (proteins sharing common ancestory)
    """

    # a bunch of temp vars to track the closest matches
    best1 = 0
    idx1 = 0
    best2 = 0
    idx2 = 0
    total = len(seqs)

    for idx in range(total):
        # Progress UI
        print(f"Running for {idx} / {total}")
        # do not match against itself - not a putative homolog
        if idx != my_sequence_number:
            seq = seqs[idx]
            # align my sequence and the current one and score
            aln = align(seqs[my_sequence_number], seq, sub_matrix, -7)
            percent = scoreAlignment(aln) / aln.alignlen

            # if it is the best, save it.
            # this means the current best is demoted the second best
            if best1 < percent:
                best2 = best1
                idx2 = idx1
                best1 = percent
                idx1 = idx

            # new second best, replace existing one
            elif best2 < percent:
                best2 = percent
                idx2 = idx

    print(f"Best match index is is {idx1}. Sequence is:\n\n{seqs[idx1]}")
    print(f"\n\nSecond best match index is is {idx2}. Sequence is:\n\n{seqs[idx2]}")
```

4.3 Part C

Links:

1. https://www.uniprot.org/uniprotkb/B1JGH2/entry
2. https://www.uniprot.org/uniprotkb/A4TM73/entry
3. https://www.uniprot.org/uniprotkb/A9R7V8/entry

All three sequences are for an identical protein. It is the (`mnmC`) gene. One comes from Yersinia pseudotuberculosis (causes scarlet fever) and the other two from Yersinia pestis (causes the plague). These two organisms are closely related and it's not surprisingly they share a gene like (`mnmC`).

It is involved in biosynthesis of 5-methylaminomethyl-2-thiouridine, in the wobble position in tRNA. The wobble position is the third position in the anti-codon and has a high amount of flexbility (it can "wobble") when pairing during translation. This means it is involved in recognising codons during translation.

5 Exercise 5: Determine the consensus sequence for the D-amino-acid oxidase MSA

Here is my code.

```
def getConsensusForColumn(aln, colidx):
    # map of count of each symbol
    symcnt = {}
    for seq in aln.seqs:
        mysym = seq[colidx]
        if mysym in symcnt:
            # increment if already found
            symcnt[mysym] += 1
        else:
            # add to map if first time encountering
            symcnt[mysym] = 1
    consensus = ""
    maxcnt = 0
    # check each symbol and select the most frequently occurring
    for mysym in symcnt:
        if symcnt[mysym] > maxcnt:
            maxcnt = symcnt[mysym]
            consensus = mysym
    # return consensus char for this column
    return consensus


def exercise_5(aln):
    # get length of the sequence
    cols = len(aln[0])
    con_seq = ""
    # loop each col and grab the next char
    for i in range(cols):
        char = getConsensusForColumn(aln, i)
        # append it
        con_seq += char
    # return a Sequence instance
    return Sequence(
```

```
        con_seq, Protein_Alphabet, name="Consensus Sequence", gappy=True)
```

The sequence:

----------SIQPATL----EWNE---D----GTPVSRQFDDVYFSNDNGLE
ETRYVFLGGNGLPERWAEH----------RLFVIAETGFGTGLNFLALWQAF
RQFRPA----------LQRLHFISFEKFPLTRADLARAHQ----------
---------HW---------P----ELAP---LAEQLLAQWP---A-LL----PGCHRLLFDEGR
VTLDLWFGDINELLPQ----------LNARVDAWFLDGFAPAKNPD-----
--MWTP----------NL------FNAMARLARPGATLATF-----TSAGFV
----------R--RGLQEAGFTVQKVK----------GFGRKREMLCGEME
QRL----------PWF-RP----------KRDAAIIGGGIAGAALAL
ALARRGWQVTLYCADEAP-AQGASGNPQGALYPLLSKDDNALSRFFRA
AFLFARRFYDALL----------G-AFDHDWCGVLQLAWDEKSAERLAKML
---ALGLPA----------ELASALDAEEAEQL---AGLPLACGGLFYPQGGWLCP
AELTRALLALA----G---TLHYGTEVQRL--ER----------D-GW----------
---------QLLDAQG----------ASAPVVVLA--NGHQITRFS----------QTA
HLP----LYPVRGQVSHIPTT----------PAL--LKT-VLCYDGYLTPA----------
-------NG--HHC-----IGASYDRGDED-----TAFREADQQ----ENLQRLQECLPD
--------W----------EVD----------VSDLQARVGVRCAT
RDHLPMVGPVPDYAATLAEYA-L----------A-PAPVYPGLYV-LGGL
G----SRGLCSAPLCAELLAAQICGEPLPLDADLLAALNPNRFWVRKLLKG
KA----------

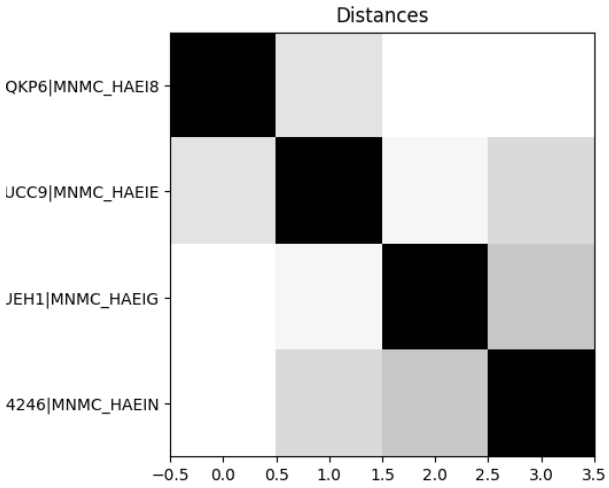
---

6 Exercise 6: Add the Poisson and Gamma distance metrics to ‘calcDistances‘

Code (see (`guide.py`) for the full code):

```
def calcDistances(self, measure = 'fractional', a=1.0):
    """ ... omitted for brevity ... """
    # Now calculate the specified measure based on p
    p = D / L
    if measure == 'fractional':
        dist = D / L
    elif measure == 'poisson':
        # This is how you calculate poisson distance. Negative log of 1 - p dist
        dist = -numpy.log(1 - p)
    elif measure == "gamma":
        # assume a = 1 for alpha
        # can be 0.2 to 3.5 according to textbook
        # todo: make it a parameter?
        # todo: why can't I use math.gamma(p)?
        # It gives drastically different numbers.
        # I guess math.gamma is not a gamma DISTANCE but just the result of gamma function?
```

```
        a = 1
        dist = a * (((1 - p) ** (-1 / a)) - 1)
    else:
        raise RuntimeError('Not implemented: %s' % measure)
```



7 Exercise 7: Curate the MalS.fa dataset

Selected 31 sequences. Code:

```
import pandas as pd


def get_yeasts():
    """Get a list of all the yeast species"""
    df = pd.read_csv("Files_WS2/sugars.csv")
    return list(df["Yeast"])


def exercise_7():
    """
    Curate the MalS.fa dataset
    """
    # Get all yeast from sugars.csv
    yeasts = get_yeasts()
    mals = readFastaFile("Files_WS2/MalS.fa", Protein_Alphabet)
    cnt = 0
    select = []
    # traverse mals dataset
    for seq in mals:
```

```
    # we are only interested in yeast also in the sugars.csv list
    if seq.name in yeasts:
        # white space -> _
        new_name = (seq.name + "_" + seq.annot).replace(" ", "_")
        seq.name = new_name
        cnt += 1
        select.append(seq)
    writeFastaFile("select.fa", select)
    print("Selected", cnt, "sequences")
```

8 Exercise 8: Identify the consensus sequence for the MalS alignment

I put my fasta file into the ClustaW web interface, and this is the consensus sequence it gave me.

---MTISSAHPETEPKWWKEATIYQIYPASFKDSN----------NDGWGDL
KGIASKLEYIKELGVDAIWICPFYDSPQDDMGYDIANYEKVWPTYGTN
EDCFALIEKTHKLGMKFITDLVINHCSSEHEWFKESRSSKTNPKRDWFF
WRPPKGYDAEGKPIPPNNWRSFFGGSAWTFDEKTQEFYLRLFASTQP
DLNWENEDCRKAIYESAVGYWLDHGVDGFRIDVGSLYSKVPGLPDAPV
TDENSKWQHSDPFTMNGPRIHEFHQEMNKFMRNRV-KDGREIMTVGE
VQHGSDETKRLYTSASRHELSELFNFSHTDVGTSPKFRYNLVPFELKDW
KVALAELFRFINGTDCWSTIYLENHDQPRSITRFGDDSPKNRVISGKLLS
VLLVSLTGTLYVYQGQELGQINF-KNWPIEKYEDVEVRNNYKAIKEEHG
ENSK---EMKKFLEGIALISRDHARTPMPWTKEEPNAGFSG---PDAKPWF
YLNESFREGINAEDESKDPNSVLNFWKEALQFRKAHKDITVYGYDFEFI
DLDNKKLFSFTKK--Y-DNKTLFAALNFSSDEIDFTIPNDSASFKLEFGNY
PDKEVDASSRTLKPWEGRIYISE--

9 Exercise 9: Locate the MalS active site in the alignment

I grabbed the sequence from the Voordeckers et al. alignment. It is for S cerevisiae S288c IMA1. Here it is:

MTISSA---HPETEPKWWKEATFYQIYPASF----------KDSNDDG-WGD
MKGIASKLEYIKELGADAIWISPFYDSPQDDMGYDIANYEKVWPTYGT
NEDCFALIEKTHKLGMKFITDLVINHCSSEHEWFKESRSSKTNPKRDWF
FWRPPKGYDAE-GKPIPPNNWKSYFGGSAWTFDEKTQEFYLRLFCSTQ
PDLNWENEDCRKAIYESAVGYWLDHGVDGFRIDVGSLYSKVVGLPDAP
VVDKNSTWQSSDPYTLNGPRIHEFHQEMNQFIRNRVK-DGREIMTVGE
MQHASD-ETKRLYTSASRHELSELFNFSHTDVGTSPLFRYNLVPFELKD
WKIALAELFRYI------NGTDCWSTIYLENHDQPRSITRFGDD-SPK-NRVIS
GKLLSVLLSALTGTLYVYQGQELGQINF-KNWPVEKYEDVEIRNNYNAI
KEEHGEN---SEEMKKFLEAIALISRDHARTPMQWS---REEPNAGFSG---P

SAKPWFYLNDSFREGINVEDEIKDPNSVLNFWKEALKFRKAHKDITVY
GYDFEFI-DLDNKKLFSFTKKYNN----KTLFAALNFSSDATDFKIPNDDSS
FKLEFGNYPKKEVDASSRTLKPWEGRIYISE----

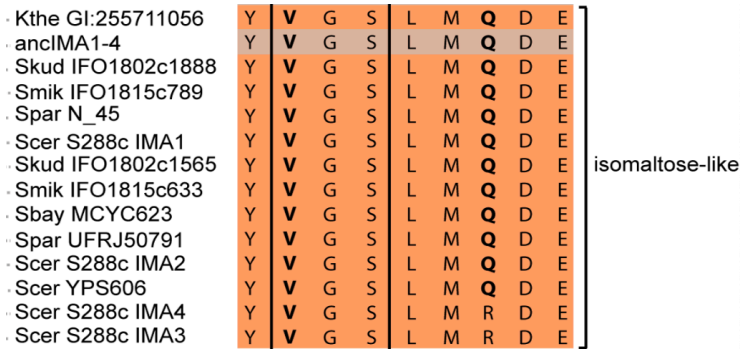
I wrote some code to verify the conserved residues match the figure in the paper.

```
voordeckers = [173, 231, 232, 233, 234, 294, 295, 324, 437]


def exercise_9():
    """
    This code print: YVGSLMQDE. This aligns with the figure 4 in the paper.
    """
    seq = "MVSMPN ... omitted for brevity ... NESA--"
    for a in voordeckers:
        print(seq[a - 1], end="")
```

Should be "YVGSLMQDE" as per figure 4 in the paper. This looks correct for the isomaltose-like group.

- 173: Y
- 231-234: VGSL
- 294-295: MQ
- 324: D
- 437: E

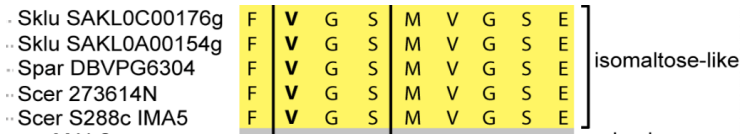


Looking at mine, I found positions:

- 173: F
- 230-234: VGSL
- 290-291: VG

- 326: S
- 437: E

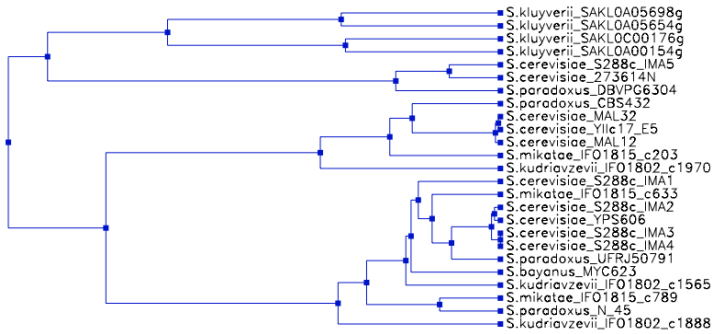
FVGSLVGSE matches the other isomaltose-like group:



10 Exercise 10: Infer the phylogenetic relationships among select MalS proteins

```
def exercise_10():
    aln = readClustalFile("exercise7.aln", Protein_Alphabet)
    phy = runUPGMA(aln, "poisson")
    print(phy)  # print so we can copy into jstree
    return phy
```

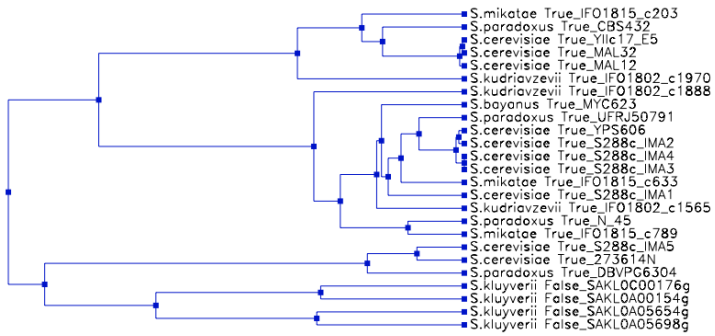
Here's the tree:



11 Exercise 11: Annotate your tree with metabolised sugars

My code is below. I loop each sugar, then for each one, loop each yeast and update the labels with string replacement.

I found Melizitose can metabolize => 8 / 14 sugars. I visualized it.



This tree suggests that at some point in the past, there was a speciation event and S.kluyverii diverged from the other Saccharomyces. At this split, S.kluyverii underwent a mutation(s) in a gene(s) and became unable to metabolize Melizitose.

The code:

```
def exercise_11():
    """Annotate tree with metabolized sugars"""
    df = pd.read_csv("Files_WS2/sugars.csv")
    yeasts = list(df["Yeast"])
    aln = readClustalFile("exercise7.aln", Protein_Alphabet)
    phy = runUPGMA(aln, "poisson")
    df = df.set_index(df["Yeast"])

    for col_name in df.columns[1:]:
        phy_copy = str(phy)
        t = 0
        f = 0
        for yeast in yeasts:
            can_metabolize = df[col_name][yeast]
            # add the labels using string replacement
            phy_copy = phy_copy.replace(yeast, f"{yeast} {can_metabolize}")
            # debugging - can see how many sugars can be metabolized
            if can_metabolize:
                t += 1
            else:
                f += 1

        # Prints: <Melizitose> => 8 / 14 = 57.14285714285714
        # Useful for debugging
        print(f"{col_name} => {t} / {f + t} = {(t / (t + f)) * 100}")
```

12 Exercise 12: Identify the active sites at inferred ancestral sequences

The residues I identified earlier are:

- 173: F
- 230-234: VGSL
- 290-291: VG
- 326: S
- 437: E

I added a (`strSites`) function, and hacked into (`_printSequences`) to get this working. I commented the interesting part.

```
def strSites(self, cols):
    return self.root._printSequences(cols)

def _printSequences(self, cols):
    """ Returns string with node (incl descendants) in a Newick style. """
    left = right = label = dist = ''
    if self.left:
        left = self.left._printSequences(cols)
    if self.right:
        right = self.right._printSequences(cols)
    if self.dist:
        dist = ':' + str(self.dist)
    if self.sequence != None:
        """
        THIS IS THE IMPORTANT PART
        USE A LIST COMPREHENSION TO MODIFY THE LABEL!
        Note: need to -1 because python starts at index=0
        """
        label = "".join(self.sequence[i-1] for i in cols) + ""
        if not self.left and not self.right:
            return label + dist
        else:
            return '(' + left + ',' + right + ')' + label + dist
    else:  # there is no label
        if not self.left and self.right:
            return ',' + right
        elif self.left and not self.right:
            return left + ','
        elif self.left and self.right:
            return '(' + left + ',' + right + ')' + dist

""" Usage """
def exercise_12():
    residues = [173, 230, 231, 232, 233, 290, 291, 326, 437]

    aln = readClustalFile("exercise7.aln", Protein_wGAP)
    tree = runUPGMA(aln, "poisson")
    tree.putAlignment(aln)
```

```
tree.parsimony()
s = tree.strSites(residues)
print(s)
```

Here is how the tree looks: