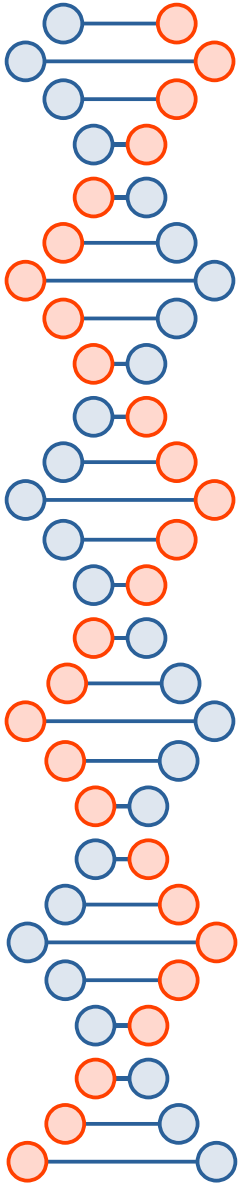

BioPython

Tools for biological computation
(https://biopython.org/wiki/Documentation)

Install BioPython

pip install biopython

- Run the above command in terminal
- more info on https://biopython.org/wiki/Download

Download Fasta

```
# Set your email before using Entrez
# Entrez.email = "your_email@example.com"   # will give some warning if email not give, but will work in both cases
# Search for the gene F56F11.4 in NCBI nucleotide database
handle = Entrez.esearch(db="nucleotide", term="F56F11.4[Gene] AND Caenorhabditis elegans[Organism]")
record = Entrez.read(handle)
handle.close()
# Get the first hit (you can loop if multiple hits)
gene_id = record["IdList"][0]
print("NCBI ID:", gene_id)
# Fetch the fasta sequence
fetch_handle = Entrez.efetch(db="nucleotide", id=gene_id, rettype="fasta", retmode="text")
seq_record = SeqIO.read(fetch_handle, "fasta")
fetch_handle.close()
# Print info
print("ID:", seq_record.id)
print("Description:", seq_record.description)
print("Sequence length:", len(seq_record.seq))
# Save to file
with open("F56F11.4.fasta", "w") as f:
    SeqIO.write(seq_record, f, "fasta")
```

Shorter version, to download gene's fasta

```
from Bio import Entrez, SeqIO; Entrez.email="you@example.com"
record = SeqIO.read(Entrez.efetch(db="nucleotide",
id=Entrez.read(Entrez.esearch(db="nucleotide", term="F56F11.4"))
["IdList"][0], rettype="fasta", retmode="text"), "fasta")
print(record.id, record.seq[:100], "...")
```

Read Fasta sequence

```
from Bio import SeqIO

# Read a FASTA file
fasta_file = "example.fasta"

for record in SeqIO.parse(fasta_file, "fasta"):
    print("ID:", record.id)
    print("Description:", record.description)
    print("Sequence:", str(record.seq))
```

Write to a file using FASTA format

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

seq = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
record = SeqRecord(seq, id="MyGene", description="Example sequence")

SeqIO.write(record, "mygene.fasta", "fasta")
```

Check nucleotide content (aacount, basecount, codoncount, dimercount, nmercount, ntdensity)

```
# 1. Plot nucleotide densities (like ntdensity)
def ntdensity(seq, window=200):
    seq = seq.upper()
    x = [i+window//2 for i in range(0, len(seq)-window, window)]
    for nt in "ATGC":
        y = [window_seq.count(nt)/window for window_seq in [seq[i:i+window] for i in range(0, len(seq)-window, window)]]
        plt.plot(x, y, label=nt)
    plt.legend(); plt.xlabel("Position"); plt.ylabel("Fraction"); plt.title("Nucleotide Density"); plt.show()

# 2. Count monomers (basecount)
def basecount(seq):
    seq = seq.upper()
    return Counter(seq)

# 3. Basecount of reverse complement
def basecount_rc(seq):
    return basecount(Seq(seq).reverse_complement())

# 4. basecount with chart (pie)
def basecount_pie(seq):
    counts = basecount(seq)
    plt.pie(counts.values(), labels=counts.keys(), autopct="%1.1f%%")
    plt.title("Nucleotide Distribution"); plt.show()

# 5. Dimer count with bar chart (dimercount)
def dimercount(seq, chart=True):
    seq = seq.upper()
    dimers = [seq[i:i+2] for i in range(len(seq)-1)]
    counts = Counter(dimers)
    if chart:
        plt.bar(counts.keys(), counts.values())
        plt.title("Dimer Counts"); plt.xlabel("Dimers"); plt.ylabel("Count")
        plt.show()
    return counts
```

Call functions to check nucleotide content

```
record = SeqIO.read("F56F11.4.fasta", "fasta")
sequence = str(record.seq)

ntdensity(sequence)                # 1
print(basecount(sequence))         # 2
print(basecount_rc(sequence))      # 3
basecount_pie(sequence)            # 4
print(dimercount(sequence))        # 5
```

Determining Codon Composition

```
from Bio.Seq import Seq
from collections import Counter
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# All 64 codons in lexicographic order
bases = ["T", "C", "A", "G"]
all_codons = [a+b+c for a in bases for b in bases for c in bases]

def codoncount(seq, frame=1, reverse=False, plot_ax=None):
    """Count codons in given frame (1,2,3) and optionally reverse strand."""
    seq = Seq(str(seq).upper())
    if reverse:
        seq = seq.reverse_complement()
    seq = seq[frame-1:]  # adjust for reading frame

    codons = [str(seq[i:i+3]) for i in range(0, len(seq)-2, 3)]
    counts = Counter(codons)

    # Fill missing codons with 0
    codon_vector = [counts.get(c, 0) for c in all_codons]
    codon_matrix = np.array(codon_vector).reshape(4,16)  # 4x16 heatmap like MATLAB

    # Plot on provided axis
    if plot_ax is not None:
        sns.heatmap(codon_matrix, ax=plot_ax,
                    xticklabels=all_codons[0:16],
                    yticklabels=["T","C","A","G"],
                    cmap="viridis", cbar=False)
        plot_ax.set_xlabel("Codons")
        plot_ax.set_ylabel("First base")

    return counts
```

Determining Codon Composition

```
# Example usage: plot all 6 reading frames
for frame in [1,2,3]:
    fig, axes = plt.subplots(2, 1, figsize=(12,6))

    codoncount(sequence, frame=frame, reverse=False, plot_ax=axes[0])
    axes[0].set_title(f"Codons for frame {frame}")

    codoncount(sequence, frame=frame, reverse=True, plot_ax=axes[1])
    axes[1].set_title(f"Codons for reverse frame {frame}")

    plt.tight_layout()
    plt.show()
```

Sequence Manipulation

```
from Bio.Seq import Seq

# --- Conversions between nucleotides, amino acids, ints ---

# 1. Amino acid -> integer mapping (1-based like MATLAB convention, A=1,...,Y=20)
aa_list = list("ARNDCQEGHILKMFPSTWYV")  # 20 standard aa
aa2int_map = {aa: i+1 for i, aa in enumerate(aa_list)}
int2aa_map = {i+1: aa for i, aa in enumerate(aa_list)}

def aa2int(aa):
    return [aa2int_map.get(a, 0) for a in str(aa).upper()]

def int2aa(ints):
    return ''.join(int2aa_map.get(i, 'X') for i in ints)

# 2. Nucleotide <-> integer (A=1, C=2, G=3, T/U=4)
nt_list = ["A", "C", "G", "T"]
nt2int_map = {nt: i+1 for i, nt in enumerate(nt_list)}
int2nt_map = {i+1: nt for i, nt in enumerate(nt_list)}

def nt2int(seq):
    return [nt2int_map.get(n, 0) for n in str(seq).upper().replace("U","T")]

def int2nt(ints):
    return ''.join(int2nt_map.get(i, 'N') for i in ints)

# 3. Codon <-> amino acid (translate DNA/RNA to protein)
def nt2aa(seq, table=1):
    return str(Seq(str(seq)).translate(table=table))

def aa2nt(aa, codon_table=1):
    # Pick the first codon for each amino acid from translation table
    from Bio.Data import CodonTable
    table = CodonTable.unambiguous_dna_by_id[codon_table]
    aa = str(aa).upper()
    seq = []
    for a in aa:
        if a in table.forward_table.values():
            codon = [k for k,v in table.forward_table.items() if v==a][0]
            seq.append(codon)
        elif a == "*":
            seq.append(table.stop_codons[0])
        else:
            seq.append("NNN")
    return ''.join(seq)
```

```
# 4. DNA <-> RNA
def dna2rna(seq):
    return str(seq).upper().replace("T","U")

def rna2dna(seq):
    return str(seq).upper().replace("U","T")

# 5. Sequence operations
def seqcomplement(seq):
    return str(Seq(str(seq)).complement())

def seqrcomplement(seq):
    return str(Seq(str(seq)).reverse_complement())

def seqreverse(seq):
    return str(seq)[::-1]
```

```
print(aa2int("MKT"))        # [13, 12, 17]
print(int2aa([13,12,17]))   # "MKT"

print(nt2int("ATGC"))       # [1,4,3,2]
print(int2nt([1,4,3,2]))    # "ATGC"

print(nt2aa("ATGGCC"))      # "MA"
print(aa2nt("MA"))          # "ATGGCC"

print(dna2rna("ATGC"))      # "AUGC"
print(rna2dna("AUGC"))      # "ATGC"

print(seqcomplement("ATGC"))    # "TACG"
print(seqrcomplement("ATGC"))   # "GCAT"
print(seqreverse("ATGC"))       # "CGTA"
```

Atomic composition

```
from collections import Counter

# Atomic composition of amino acids (monoisotopic, no terminal modifications)
aa_atoms = {
    "A": {"C":3,"H":7,"N":1,"O":2},   # Alanine
    "R": {"C":6,"H":14,"N":4,"O":2},  # Arginine
    "N": {"C":4,"H":8,"N":2,"O":3},   # Asparagine
    "D": {"C":4,"H":7,"N":1,"O":4},   # Aspartic Acid
    "C": {"C":3,"H":7,"N":1,"O":2,"S":1},
    "E": {"C":5,"H":9,"N":1,"O":4},
    "Q": {"C":5,"H":10,"N":2,"O":3},
    "G": {"C":2,"H":5,"N":1,"O":2},
    "H": {"C":6,"H":9,"N":3,"O":2},
    "I": {"C":6,"H":13,"N":1,"O":2},
    "L": {"C":6,"H":13,"N":1,"O":2},
    "K": {"C":6,"H":14,"N":2,"O":2},
    "M": {"C":5,"H":11,"N":1,"O":2,"S":1},
    "F": {"C":9,"H":11,"N":1,"O":2},
    "P": {"C":5,"H":9,"N":1,"O":2},
    "S": {"C":3,"H":7,"N":1,"O":3},
    "T": {"C":4,"H":9,"N":1,"O":3},
    "W": {"C":11,"H":12,"N":2,"O":2},
    "Y": {"C":9,"H":11,"N":1,"O":3},
    "V": {"C":5,"H":11,"N":1,"O":2}
}

def atomic_composition(seq):
    comp = Counter()
    for aa in seq:
        if aa in aa_atoms:
            comp.update(aa_atoms[aa])
    # Adjust for peptide bond formation: each bond removes H2O
    n_bonds = len(seq)-1
    comp["H"] -= 2*n_bonds
    comp["O"] -= n_bonds
    return dict(comp)

# Example: Human HEXA (short fragment)
seq = "MVLTIYPDELVQIVSDKK"
print(atomic_composition(seq))
```

Molecular weight

```
from Bio.SeqUtils import molecular_weight

seq = "MVLTIYPDELVQIVSDKK"
mw = molecular_weight(seq, seq_type="protein")
print(mw)
# You can also set monoisotopic=True if you want exact monoisotopic weight instead of average:
mw_mono = molecular_weight(seq, seq_type="protein", monoisotopic=True)
print(mw_mono)
```

Protein Analysis

```
from Bio.SeqUtils.ProtParam import ProteinAnalysis

seq = "MVLTIYPDELVQIVSDKK"
prot = ProteinAnalysis(seq)

print("Length:", len(seq))
print("Molecular Weight:", prot.molecular_weight())
print("Aromaticity:", prot.aromaticity())
print("Instability Index:", prot.instability_index())
print("Isoelectric Point (pI):", prot.isoelectric_point())
print("GRAVY (hydropathy):", prot.gravy())
print("Amino Acid Percent Composition:", prot.get_amino_acids_percent())
```

Global Alignment (Needleman-Wunsch)

```
from Bio import pairwise2
from Bio.pairwise2 import format_alignment

seq1 = "GATTACA"
seq2 = "GCATGCU"

# Global alignment (Needleman-Wunsch)
alignments = pairwise2.align.globalxx(seq1, seq2)

# Print top alignment
print(format_alignment(*alignments[0]))
```

Local alignment (Smith-Waterman)

```
from Bio import pairwise2
from Bio.pairwise2 import format_alignment

seq1 = "GATTACA"
seq2 = "GCATGCU"

# Local alignment (Smith-Waterman)
alignments = pairwise2.align.localxx(seq1, seq2)

# Print top alignment
print(format_alignment(*alignments[0]))
```

Retrieve pdb file

```
from Bio.PDB import PDBList

pdbl = PDBList()
pdbl.retrieve_pdb_file("1A3N", file_format="pdb",
pdir=".")
```