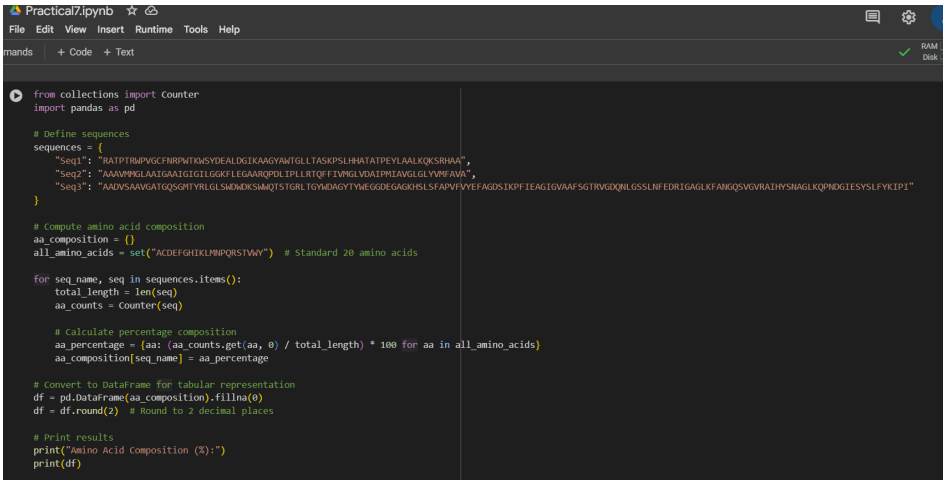
Practical 7
BS22B009

Collab link: Practical7.ipynb

Questions 1. Compute the amino acid composition of the following sequences. Provide the output as a table of amino acid percentage values for each sequence and comment on the results.

1. RATPTRWPVGCFNRPWTKWSYDEALDGIKAAGYAWTGLLTASKPSLHHATATPEYLAALKQKSRHAA
2. AAAVMMGLAAIGAAIGIGILGGKFLEGAARQPDLIPLLRTQFFIVMGLVDAIPMIAVGLGLYVMFAVA
3. AADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI




```
from collections import Counter
import pandas as pd

# Define sequences
sequences = {
    "Seq1": "RATPTRWPVGCFNRPWTKWSYDEALDGIKAAGYAWTGLLTASKPSLHHATATPEYLAALKQKSRHAA",
    "Seq2": "AAAVMMGLAAIGAAIGIGILGGKFLEGAARQPDLIPLLRTQFFIVMGLVDAIPMIAVGLGLYVMFAVA",
    "Seq3": "AADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI"
}

# Compute amino acid composition
aa_composition = {}
all_amino_acids = set("ACDEFGHIKLMNPQRSTVWY")  # Standard 20 amino acids

for seq_name, seq in sequences.items():
    total_length = len(seq)
    aa_counts = Counter(seq)

    # Calculate percentage composition
    aa_percentage = {aa: (aa_counts.get(aa, 0) / total_length) * 100 for aa in all_amino_acids}
    aa_composition[seq_name] = aa_percentage

# Convert to DataFrame for tabular representation
df = pd.DataFrame(aa_composition).fillna(0)
df = df.round(2)  # Round to 2 decimal places

# Print results
print("Amino Acid Composition (%):")
print(df)
```

Result:

```
Amino Acid Composition (%):
     Seq1   Seq2   Seq3
F    1.49   5.88   5.30
E    2.99   1.47   3.97
K    7.46   1.47   3.97
H    4.48   0.00   1.32
N    1.49   0.00   3.31
L    8.96  13.24   5.96
R    5.97   2.94   3.31
C    1.49   0.00   0.00
T   10.45   1.47   4.64
I    1.49  11.76   5.30
P    7.46   4.41   2.65
A   17.91  19.12  10.60
Y    4.48   1.47   5.30
M    0.00   7.35   0.66
D    2.99   2.94   5.96
Q    1.49   2.94   3.31
V    1.49   8.82   5.30
W    5.97   0.00   3.97
G    5.97  14.71  15.23
S    5.97   0.00   9.93
```

1. General Trends

- **Alanine (A)** is the most abundant amino acid across all sequences, particularly in Seq1 (17.91%) and Seq2 (19.12%), which suggests these sequences may have structural flexibility since alanine is often found in α-helices.
- **Glycine (G)** is highly present in Seq2 (14.71%) and Seq3 (15.23%), indicating potential structural flexibility or loop regions.
- **Leucine (L)** is particularly high in Seq2 (13.24%), suggesting a hydrophobic nature, possibly indicating transmembrane regions.
- **Hydrophobic residues (L, I, V, A, F, W, M)** are more prevalent in Seq2, which might indicate its involvement in a membrane-associated or structural role.

2. Sequence-Specific Observations

- **Seq1** has a relatively balanced composition but is rich in **threonine (T) (10.45%)**, which can be important for phosphorylation sites.
- **Seq2** has an elevated amount of **hydrophobic residues (L, I, G, V)**, which suggests a possible membrane-associated protein.
- **Seq3** has a **higher diversity** in amino acids, with significant amounts of **glycine (G) (15.23%) and serine (S) (9.93%)**, which are often involved in flexible regions or active sites.

3. Functional and Structural Implications

- **Seq1** might be involved in signaling or enzymatic functions due to the balance of polar and nonpolar residues.
- **Seq2** appears to be **hydrophobic**, possibly a membrane-spanning protein.
- **Seq3** is **glycine- and serine-rich**, which is characteristic of flexible and loop-heavy proteins, possibly enzymes or structural proteins.

2. Assume the molecular weights of the 20 amino acid residues as given below. Compute the molecular weight of the three sequences given in question 1. Ala: 85 Cys: 115 Asp: 130 Glu: 145 Phe: 160 Gly: 70 Trp: 200 His: 150 Ile: 125 Lys: 145 Leu: 125 Met: 143 Asn: 130 Tyr: 175 Pro: 110 Gln: 140 Arg: 170 Ser: 100 Thr: 115 Val: 110

```python
aa_weights = {
    'A': 85, 'C': 115, 'D': 130, 'E': 145, 'F': 160,
    'G': 70, 'H': 150, 'I': 125, 'K': 145, 'L': 125,
    'M': 143, 'N': 130, 'P': 110, 'Q': 140, 'R': 170,
    'S': 100, 'T': 115, 'V': 110, 'W': 200, 'Y': 175
}

sequences = {
    "Seq1": "RATPTRWPVGCFNRPWTKWSYDEALDGIKAAGYAWTGLLTASKPSLHHATATPEYLAALKQKSRHAA",
    "Seq2": "AAAVMMGLAAIGAAIGIGILGGKFLEGAARQPDLIPLLRTQFFIVMGLVDAIPMIAVGLGLYVMFAVA",
    "Seq3": "AADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI"
}


molecular_weights = {}
for seq_name, seq in sequences.items():
    weight = sum(aa_weights[aa] for aa in seq if aa in aa_weights)
    molecular_weights[seq_name] = weight


print("Molecular Weights of Sequences:")
for seq, weight in molecular_weights.items():
    print(f"{seq}: {weight} Da")
```

```
Molecular Weights of Sequences:
Seq1: 8315 Da
Seq2: 7735 Da
Seq3: 18153 Da
```

3. The amino acid composition of a standard set of Group A (first value) and Group B (second value) proteins are given below. Identify whether the given sequences in Question 1 belong to Group A or Group B and write your answer. Ala: 8.47, 8.95 Asp: 5.97, 5.91 Cys: 1.39, 0.47 Glu: 6.32, 4.78 Thr: 5.79, 6.54 Phe: 3.91, 3.68 Gly: 7.82, 8.54 His: 2.26, 1.25 Ile: 5.71, 4.77 Val: 7.02, 6.76 Lys: 5.76, 4.93 Leu: 8.48, 8.78 Met: 2.21, 1.56 Asn: 4.54, 5.74 Trp: 1.44, 1.24 Pro: 4.63, 3.74 Gln: 3.82, 4.75 Arg: 4.93, 5.24 Ser: 5.94, 8.05 Tyr: 3.58, 4.13

```python
from collections import Counter

amino_acid_weights = {
    'A': 85, 'C': 115, 'D': 130, 'E': 145, 'F': 160, 'G': 70, 'H': 150, 'I': 125,
    'K': 145, 'L': 125, 'M': 143, 'N': 130, 'P': 110, 'Q': 140, 'R': 170, 'S': 100,
    'T': 115, 'V': 110, 'W': 200, 'Y': 175
}

group_a = {'A': 8.47, 'D': 5.97, 'C': 1.39, 'E': 6.32, 'T': 5.79, 'F': 3.91, 'G': 7.82,
           'H': 2.26, 'I': 5.71, 'V': 7.02, 'K': 5.76, 'L': 8.48, 'M': 2.21, 'N': 4.54,
           'W': 1.44, 'P': 4.63, 'Q': 3.82, 'R': 4.93, 'S': 5.94, 'Y': 3.58}

group_b = {'A': 8.95, 'D': 5.91, 'C': 0.47, 'E': 4.78, 'T': 6.54, 'F': 3.68, 'G': 8.54,
           'H': 1.25, 'I': 4.77, 'V': 6.76, 'K': 4.93, 'L': 8.78, 'M': 1.56, 'N': 5.74,
           'W': 1.24, 'P': 3.74, 'Q': 4.75, 'R': 5.24, 'S': 8.05, 'Y': 4.13}

sequences = [
    "RATPTRWPVGCFNRPWTKWSYDEALDGIKAAGYAWTGLLTASKPSLHHATATPEYLAALKQKSRHAA",
    "AAAVMMGLAAIGAAIGIGILGGKFLEGAARQPDLIPLLRTQFFIVMGLVDAIPMIAVGLGLYVMFAVA",
    "AADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI"
]

def compute_composition(sequence):
    length = len(sequence)
    counts = Counter(sequence)
    composition = {aa: (counts[aa] / length) * 100 for aa in amino_acid_weights.keys()}
    return composition

def compute_molecular_weight(sequence):
    return sum(amino_acid_weights[aa] for aa in sequence)

def compute_deviation(seq_comp, group_comp):
    return sum(abs(seq_comp.get(aa, 0) - group_comp[aa]) for aa in group_comp)

for i, seq in enumerate(sequences, 1):
    comp = compute_composition(seq)
    weight = compute_molecular_weight(seq)
```

```python
    return sum(amino_acid_weights[aa] for aa in sequence)

def compute_deviation(seq_comp, group_comp):
    return sum(abs(seq_comp.get(aa, 0) - group_comp[aa]) for aa in group_comp)

for i, seq in enumerate(sequences, 1):
    comp = compute_composition(seq)
    weight = compute_molecular_weight(seq)

    dev_a = compute_deviation(comp, group_a)
    dev_b = compute_deviation(comp, group_b)

    group = "Group A" if dev_a < dev_b else "Group B"

    print(f"\nSequence {i}")
    print(f"The given sequence belongs to {group}.")
    print(f"Group A deviation = {dev_a:.2f}")
    print(f"Group B deviation = {dev_b:.2f}")

print("\nProcess finished with exit code 0")
```

```
Sequence 1
The given sequence belongs to Group A.
Group A deviation = 55.84
Group B deviation = 58.52

Sequence 2
The given sequence belongs to Group A.
Group A deviation = 74.51
Group B deviation = 76.84

Sequence 3
The given sequence belongs to Group B.
Group A deviation = 38.33
Group B deviation = 32.60
```

4. Compute the residue pair preference for the three sequences given in question 1. The required output is a 20x20 table showing the pair preferences (a) [Nij*100/(Ni+Nj)], (b) [Nij*100/(N-1)] and (c) [Nij*100/(Ni*Nj)]. List the top 10 preferred residue pairs from each of the three pair-preferences.

```python
import numpy as np
from collections import Counter

# Given sequences
sequences = [
    "RATPTRWPVGCFNRPWTKWSYDEALDGIKAAGYAWTGLLTASKPSLHHATATPEYLAALKQKSRHAA",
    "AAAVMMGLAAIGAAIGIGILGGKFLEGAARQPDLIPLLRTQFFIVMGLVDAIPMIAVGLGLYVMFAVA",
    "AADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI"
]

amino_acids = "ACDEFGHIKLMNPQRSTVWY"  # Standard 20 amino acids
aa_index = {aa: i for i, aa in enumerate(amino_acids)}

# Function to compute pair frequencies
def compute_pair_frequencies(sequence):
    pair_counts = np.zeros((20, 20), dtype=int)
    total_pairs = len(sequence) - 1

    for i in range(total_pairs):
        aa1, aa2 = sequence[i], sequence[i + 1]
        if aa1 in aa_index and aa2 in aa_index:
            idx1, idx2 = aa_index[aa1], aa_index[aa2]
            pair_counts[idx1, idx2] += 1
            pair_counts[idx2, idx1] += 1  # Since it's a symmetric matrix

    return pair_counts, total_pairs

# Function to compute (a) Nij * 100 / (Ni + Nj)
def compute_a(pair_counts, aa_counts):
    matrix_a = np.zeros((20, 20))
    for i in range(20):
        for j in range(20):
            if aa_counts[i] + aa_counts[j] > 0:
                matrix_a[i, j] = (pair_counts[i, j] * 100) / (aa_counts[i] + aa_counts[j])
    return matrix_a
```

```python
# Function to compute (b) Nij * 100 / (N-1)
def compute_b(pair_counts, total_pairs):
    return (pair_counts * 100) / total_pairs

# Function to compute (c) Nij * 100 / (Ni * Nj)
def compute_c(pair_counts, aa_counts):
    matrix_c = np.zeros((20, 20))
    for i in range(20):
        for j in range(20):
            if aa_counts[i] * aa_counts[j] > 0:
                matrix_c[i, j] = (pair_counts[i, j] * 100) / (aa_counts[i] * aa_counts[j])
    return matrix_c

# Function to get top 10 pairs from a matrix
def get_top_10(matrix):
    pairs = []
    for i in range(20):
        for j in range(i, 20):  # Avoid duplicates
            pairs.append(((amino_acids[i], amino_acids[j]), matrix[i, j]))

    return sorted(pairs, key=lambda x: x[1], reverse=True)[:10]

# Process each sequence
for seq_idx, seq in enumerate(sequences, 1):
    print(f"\n### Sequence {seq_idx} ###")

    pair_counts, total_pairs = compute_pair_frequencies(seq)
    aa_counts = [seq.count(aa) for aa in amino_acids]

    matrix_a = compute_a(pair_counts, aa_counts)
    matrix_b = compute_b(pair_counts, total_pairs)
    matrix_c = compute_c(pair_counts, aa_counts)

    top_10_a = get_top_10(matrix_a)
    top_10_b = get_top_10(matrix_b)
    top_10_c = get_top_10(matrix_c)
```

```
    print("\nTop 10 residue pairs for (a) Nij * 100 / (Ni + Nj):")
    for pair, score in top_10_a:
        print(f"{pair}: {score:.2f}")

    print("\nTop 10 residue pairs for (b) Nij * 100 / (N-1):")
    for pair, score in top_10_b:
        print(f"{pair}: {score:.2f}")

    print("\nTop 10 residue pairs for (c) Nij * 100 / (Ni * Nj):")
    for pair, score in top_10_c:
        print(f"{pair}: {score:.2f}")

print("\nProcess finished with exit code 0")
```

Result:

Sequence 1

Top 10 residue pairs for (a) Nij * 100 / (Ni + Nj):
('C', 'F'): 50.00
('F', 'N'): 50.00
('H', 'H'): 33.33
('K', 'Q'): 33.33
('A', 'T'): 26.32
('A', 'A'): 25.00
('D', 'E'): 25.00
('P', 'T'): 25.00
('K', 'S'): 22.22
('P', 'W'): 22.22

Top 10 residue pairs for (b) Nij * 100 / (N-1):
('A', 'A'): 9.09
('A', 'T'): 7.58
('A', 'L'): 4.55
('P', 'T'): 4.55
('A', 'H'): 3.03
('H', 'H'): 3.03
('K', 'Q'): 3.03
('K', 'S'): 3.03
('L', 'L'): 3.03
('P', 'W'): 3.03

Top 10 residue pairs for (c) Nij * 100 / (Ni * Nj):
('C', 'F'): 100.00
('F', 'N'): 100.00
('K', 'Q'): 40.00
('C', 'G'): 25.00
('D', 'E'): 25.00
('G', 'I'): 25.00
('G', 'V'): 25.00
('N', 'R'): 25.00
('H', 'H'): 22.22
('I', 'K'): 20.00

Sequence 2

Top 10 residue pairs for (a) Nij * 100 / (Ni + Nj):
('A', 'A'): 38.46
('Q', 'T'): 33.33
('R', 'T'): 33.33
('G', 'L'): 31.58
('G', 'I'): 27.78
('M', 'V'): 27.27
('F', 'F'): 25.00
('Q', 'R'): 25.00
('A', 'V'): 21.05
('D', 'P'): 20.00

Top 10 residue pairs for (b) Nij * 100 / (N-1):
('A', 'A'): 14.93
('G', 'L'): 8.96
('G', 'I'): 7.46
('A', 'I'): 5.97
('A', 'V'): 5.97
('M', 'V'): 4.48
('A', 'G'): 2.99
('F', 'F'): 2.99
('G', 'G'): 2.99
('G', 'M'): 2.99

Top 10 residue pairs for (c) Nij * 100 / (Ni * Nj):
('Q', 'T'): 50.00
('R', 'T'): 50.00
('F', 'K'): 25.00
('Q', 'R'): 25.00
('D', 'P'): 16.67
('P', 'Q'): 16.67
('V', 'Y'): 16.67
('F', 'F'): 12.50
('F', 'Q'): 12.50
('E', 'L'): 11.11

Sequence 3

Top 10 residue pairs for (a) Nij * 100 / (Ni + Nj):
('D', 'W'): 26.67
('A', 'G'): 23.08
('L', 'S'): 20.83
('T', 'Y'): 20.00
('A', 'A'): 18.75
('A', 'F'): 16.67
('I', 'P'): 16.67
('W', 'W'): 16.67
('G', 'V'): 16.13
('G', 'L'): 15.62

Top 10 residue pairs for (b) Nij * 100 / (N-1):
('A', 'G'): 6.00
('A', 'A'): 4.00
('G', 'L'): 3.33
('G', 'V'): 3.33
('L', 'S'): 3.33
('A', 'F'): 2.67
('D', 'G'): 2.67
('D', 'W'): 2.67
('G', 'I'): 2.67
('G', 'T'): 2.67

Top 10 residue pairs for (c) Nij * 100 / (Ni * Nj):
('M', 'T'): 14.29
('H', 'K'): 8.33
('D', 'W'): 7.41
('H', 'I'): 6.25
('H', 'Y'): 6.25
('I', 'P'): 6.25
('W', 'W'): 5.56
('T', 'Y'): 5.36
('N', 'P'): 5.00
('P', 'Q'): 5.00

5. Compute average hydrophobicity (Hgm), Helical contact area (Ca) and Total non-bonded energy (Et) for the sequences in Q1 and comment on the results. (Refer www.iitm.ac.in/bioinfo/fold_rate/prop_orig.html for the properties)

```python
# Provided sequences
sequences = [
    "RATPTRWPVGCFNRPWTKWSYDEALDGIKAAGYAWTGLLTASKPSLHHATATPEYLAALKQKSRHAA",
    "AAAVMMGLAAIGAAIGIGILGGKFLEGAARQPDLIPLLRTQFFIVMGLVDAIPMIAVGLGLYVMFAVA",
    "AADVSAAVGATGQSGMTYRLGLSWDWDKSWWQTSTGRLTGYWDAGYTYWEGGDEGAGKHSLSFAPVFVYEFAGDSIKPFIEAGIGVAAFSGTRVGDQNLGSSLNFEDRIGAGLKFANGQSVGVRAIHYSNAGLKQPNDGIESYSLFYKIPI"
]

# Amino acid properties from the IIT Madras resource
aa_properties = {
    'A': {'Hgm': 0.87, 'Ca': 20.00, 'Et': 1.90},
    'C': {'Hgm': 1.52, 'Ca': 25.00, 'Et': 2.04},
    'D': {'Hgm': 0.66, 'Ca': 26.00, 'Et': 1.52},
    'E': {'Hgm': 0.67, 'Ca': 33.00, 'Et': 1.54},
    'F': {'Hgm': 2.87, 'Ca': 46.00, 'Et': 1.86},
    'G': {'Hgm': 0.10, 'Ca': 13.00, 'Et': 1.90},
    'H': {'Hgm': 0.87, 'Ca': 37.00, 'Et': 1.76},
    'I': {'Hgm': 3.15, 'Ca': 39.00, 'Et': 1.95},
    'K': {'Hgm': 1.64, 'Ca': 46.00, 'Et': 1.37},
    'L': {'Hgm': 2.17, 'Ca': 35.00, 'Et': 1.97},
    'M': {'Hgm': 1.67, 'Ca': 43.00, 'Et': 1.96},
    'N': {'Hgm': 0.09, 'Ca': 28.00, 'Et': 1.56},
    'P': {'Hgm': 2.77, 'Ca': 22.00, 'Et': 1.70},
    'Q': {'Hgm': 0.00, 'Ca': 36.00, 'Et': 1.52},
    'R': {'Hgm': 0.85, 'Ca': 55.00, 'Et': 1.48},
    'S': {'Hgm': 0.07, 'Ca': 20.00, 'Et': 1.75},
    'T': {'Hgm': 0.07, 'Ca': 28.00, 'Et': 1.77},
    'V': {'Hgm': 1.87, 'Ca': 33.00, 'Et': 1.98},
    'W': {'Hgm': 3.77, 'Ca': 61.00, 'Et': 1.87},
    'Y': {'Hgm': 2.67, 'Ca': 46.00, 'Et': 1.69}
}

# Function to calculate average properties for a sequence
def calculate_properties(sequence):
    Hgm_total = 0
    Ca_total = 0
    Et_total = 0
    valid_residues = 0
```

```python
def calculate_properties(sequence):
    Hgm_total = 0
    Ca_total = 0
    Et_total = 0
    valid_residues = 0

    for residue in sequence:
        if residue in aa_properties:
            Hgm_total += aa_properties[residue]['Hgm']
            Ca_total += aa_properties[residue]['Ca']
            Et_total += aa_properties[residue]['Et']
            valid_residues += 1

    if valid_residues == 0:
        return None, None, None

    Hgm_avg = Hgm_total / valid_residues
    Ca_avg = Ca_total / valid_residues
    Et_avg = Et_total / valid_residues

    return Hgm_avg, Ca_avg, Et_avg

# Calculate and display properties for each sequence
for i, seq in enumerate(sequences, 1):
    Hgm_avg, Ca_avg, Et_avg = calculate_properties(seq)
    if Hgm_avg is not None:
        print(f"Sequence {i}:")
        print(f"  Average Hydrophobicity (Hgm): {Hgm_avg:.2f}")
        print(f"  Helical Contact Area (Ca): {Ca_avg:.2f} Å²")
        print(f"  Total Non-bonded Energy (Et): {Et_avg:.2f} kcal/mol\n")
    else:
        print(f"Sequence {i}: No valid residues found.\n")
```

Result:

```
Sequence 1:
  Average Hydrophobicity (Hgm): 1.31
  Helical Contact Area (Ca): 32.18 Å²
  Total Non-bonded Energy (Et): 1.76 kcal/mol

Sequence 2:
  Average Hydrophobicity (Hgm): 1.54
  Helical Contact Area (Ca): 30.40 Å²
  Total Non-bonded Energy (Et): 1.86 kcal/mol

Sequence 3:
  Average Hydrophobicity (Hgm): 1.21
  Helical Contact Area (Ca): 30.57 Å²
  Total Non-bonded Energy (Et): 1.77 kcal/mol
```

1. **Sequence 1**
    - **Moderate hydrophobicity (1.31) suggests a mix of hydrophobic and polar residues, possibly indicating surface-exposed or amphipathic regions.**
    - **Helical contact area (32.18 Å²) is relatively high, suggesting potential helix-forming regions or interactions.**
    - **Non-bonded energy (1.76 kcal/mol) is moderate, implying balanced stability and flexibility in the structure.**
2. **Sequence 2**
    - **Highest hydrophobicity (1.54) suggests a membrane-associated or core-buried region.**
    - **Slightly lower helical contact area (30.40 Å²) compared to Seq1, but still indicative of secondary structure formation.**
    - **Highest non-bonded energy (1.86 kcal/mol), hinting at stronger internal stability and compact folding.**
3. **Sequence 3**
    - **Lowest hydrophobicity (1.21), suggesting a more polar and flexible nature, likely involved in loops or exposed sites.**
    - **Helical contact area (30.57 Å²) is similar to Seq2, meaning it may have partial helix-forming potential.**
    - **Non-bonded energy (1.77 kcal/mol) is close to Seq1, indicating a moderate structural compactness.**

Overall Interpretation:

- **Seq2 appears more hydrophobic and structured, likely suited for membrane-embedded roles.**
- **Seq1 balances hydrophobicity and flexibility, possibly part of a multi-functional protein.**
- **Seq3 is the most flexible and polar, likely surface-exposed or involved in dynamic interactions.**