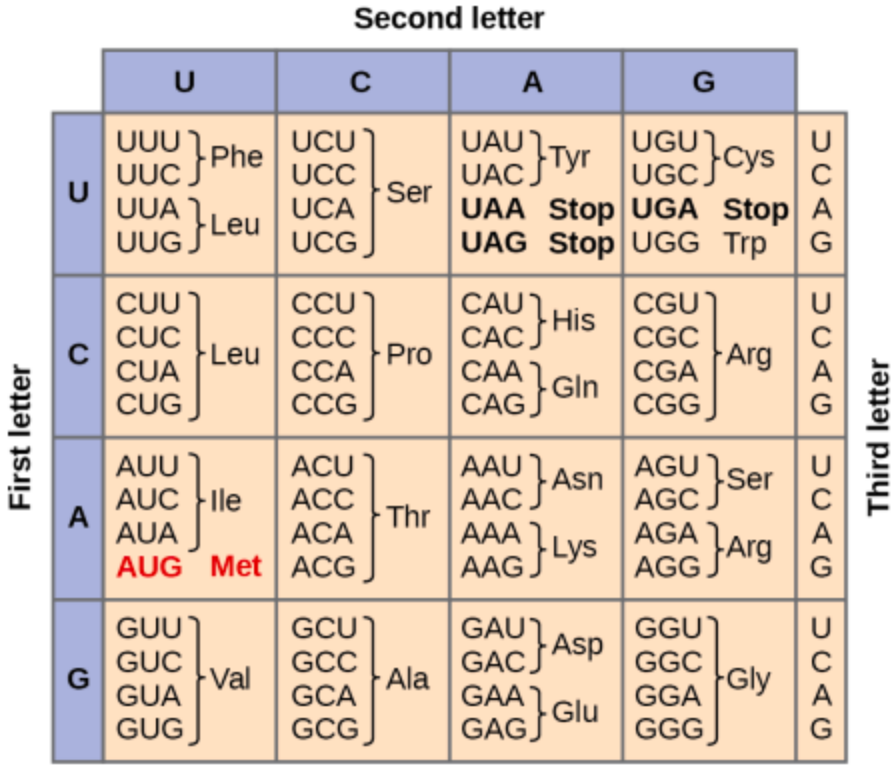
Lab Sheet 2

AKSHAI D PILLAI

AM.EN.U4AIE22102

1. Create a Python dictionary to store the RNA codon table explained in the class(Use the one letter representation of the amino acid). Download the DNA sequence of 'Insulin' from NCBI and do the process of transcription and translation to see which amino acid sequence is produced from it. Compare your result against the amino acid sequence of Insulin downloaded from NCBI.



```
import re
import random

def find_orf(dna_sequence):
    start_codon = "AUG"
    stop_codons = ["UAA", "UAG", "UGA"]
    orfs = []
    for i in range(len(dna_sequence)):
        if dna_sequence[i:i + 3] == start_codon:
            for j in range(i + 3, len(dna_sequence), 3):
                codon = dna_sequence[j:j + 3]
                if codon in stop_codons:
                    orf = dna_sequence[i:j + 3]
                    orfs.append(orf)
                    break

    return orfs

def C(dna):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    return ''.join([complement[base] for base in dna])

def Protein(f):
  rna_codon_table = {
    'UUU': 'F', 'UUC': 'F', 'UUA': 'L', 'UUG': 'L',
    'CUU': 'L', 'CUC': 'L', 'CUA': 'L', 'CUG': 'L',
    'AUU': 'I', 'AUC': 'I', 'AUA': 'I', 'AUG': 'M',
    'GUU': 'V', 'GUC': 'V', 'GUA': 'V', 'GUG': 'V',
    'UCU': 'S', 'UCC': 'S', 'UCA': 'S', 'UCG': 'S',
    'CCU': 'P', 'CCC': 'P', 'CCA': 'P', 'CCG': 'P',
    'ACU': 'T', 'ACC': 'T', 'ACA': 'T', 'ACG': 'T',
    'GCU': 'A', 'GCC': 'A', 'GCA': 'A', 'GCG': 'A',
    'UAU': 'Y', 'UAC': 'Y', 'UAA': '*', 'UAG': '*',
    'CAU': 'H', 'CAC': 'H', 'CAA': 'Q', 'CAG': 'Q',
    'AAU': 'N', 'AAC': 'N', 'AAA': 'K', 'AAG': 'K',
    'GAU': 'D', 'GAC': 'D', 'GAA': 'E', 'GAG': 'E',
    'UGU': 'C', 'UGC': 'C', 'UGA': '*', 'UGG': 'W',
    'CGU': 'R', 'CGC': 'R', 'CGA': 'R', 'CGG': 'R',
    'AGU': 'S', 'AGC': 'S', 'AGA': 'R', 'AGG': 'R',
    'GGU': 'G', 'GGC': 'G', 'GGA': 'G', 'GGG': 'G',
  }
  p = ""
  for i in range(0, len(f), 3):
      codon = f[i:i + 3]
      amino_acid = rna_codon_table.get(codon, '')
      p += amino_acid

  return p

def Frames(f,frame):
  c = 1
```

```
  protiens = []
  for i in f:
    print(f"{frame} Frame {c} : ",i)
    p = Protein(i)
    print("Protein      : ",p)
    protiens.append(p[:-1])
    c += 1
    print()
  return protiens


import re
dna = """AGCCCTCCAGGACAGGCTGCATCAGAAGAGGCCATCAAGCAGATCACTGTCCTTCTGCCATGGCCCTGTG
GATGCGCCTCCTGCCCCTGCTGGCGCTGCTGGCCCTCTGGGGACCTGACCCAGCCGCAGCCTTTGTGAAC
CAACACCTGTGCGGCTCACACCTGGTGGAAGCTCTCTACCTAGTGTGCGGGGAACGAGGCTTCTTCTACA
CACCCAAGACCCGCCGGGAGGCAGAGGACCTGCAGGTGGGGCAGGTGGAGCTGGGCGGGGGCCCTGGTGC
AGGCAGCCTGCAGCCCTTGGCCCTGGAGGGGTCCCTGCAGAAGCGTGGCATTGTGGAACAATGCTGTACC
AGCATCTGCTCCCTCTACCAGCTGGAGAACTACTGCAACTAGACGCAGCCCGCAGGCAGCCCCACACCCG
CCGCCTCCTGCACCGAGAGAGATGGAATAAAGCCCTTGAACCAGC"""

dna=re.sub('\s','',dna)

mrna = dna.replace('T','U')
print("The mRNA sequence is: ", mrna)

f = find_orf(mrna)
orginal = "MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGSLQPLA

for i in f:
  if len(i[:-3])//3==len(orginal):
    f = i[:-3]
    break

print("5'3'Frame: ",f)
protein_sequence = Protein(f)


print("\nOrginal Protein Sequence:")
print(orginal)
print("\nProtein Sequence generated:")
print(protein_sequence)

if protein_sequence == orginal:
  print("\nYes, Both are same!")
```

```
The mRNA sequence is:  AGCCCUCCAGGACAGGCUGCAUCAGAAGAGGCCAUCAAGCAGAUCACUGUCCUUCUGCCAUGGCC
5'3'Frame:  AUGGCCCUGUGGAUGCGCCUCCUGCCCCUGCUGGCGCUGCUGGCCCUCUGGGGACCUGACCCAGCCGCAGCCUUUG

Orginal Protein Sequence:
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGSLQPLALEGSLQK

Protein Sequence generated:
```

```
MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGSLQPLALEGSLQK
```

Yes, Both are same!

```
sequences = []
current_sequence = ""

with open("/content/rna.fna", "r") as file:
    for line in file:
        if line.startswith(">"):
            if current_sequence:
                sequences.append(current_sequence)
            current_sequence = ""
        else:
            current_sequence += line.strip()

if current_sequence:
    sequences.append(current_sequence)

dna = random.choice(sequences)
print("DNA                          :",dna)
mrna = dna.replace('T','U')
print("MRNA                         :",mrna)
cdna = C(dna)
print("Complimnet of DNA            :",cdna)
rcdna = cdna[::-1]
print("Reverse Complimnet of DNA    :",rcdna)
rcmrna = rcdna.replace('T','U')
print("MRNA for Reverse Complimnet :",rcmrna)
```

```
DNA                          : AGCCCTCCAGGACAGGCTGCATCAGAAGAGGCCATCAAGCAGGTCTGTTCCAAGGGCC
MRNA                         : AGCCCUCCAGGACAGGCUGCAUCAGAAGAGGCCAUCAAGCAGGUCUGUUCCAAGGGCC
Complimnet of DNA            : TCGGGAGGTCCTGTCCGACGTAGTCTTCTCCGGTAGTTCGTCCAGACAAGGTTCCCGG
Reverse Complimnet of DNA    : GCTGGTTCAAGGGCTTTATTCCATCTCTCTCGGTGCAGGAGGCGGCGGGTGTGGGGCT
MRNA for Reverse Complimnet : GCUGGUUCAAGGGCUUUAUUCCAUCUCUCUCGGUGCAGGAGGCGGCGGGUGUGGGGCU
```

```
f53 = find_orf(mrna)
p = Frames(f53,"5'3'")
f35 = find_orf(rcmrna)
p += Frames(f35,"3'5'")

orginal = "MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGSLQPLA
generated = ""
for i in p:
  if len(i)==len(orginal):
    generated = i
    break

print("Generated Protein Sequence :",generated)
print("Orginal Protein Sequence   :",orginal)
if orginal == generated:
  print("Both are same !")
else:
  print("Both are not same !")
```

```
5'3' Frame 1 :  AUGGCCCUGUGGAUGCGCCUCCUGCCCCUGCUGGCGCUGCUGGCCCUCUGGGGACCUGACCCAGCCGCAGCC
Protein      :  MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGP

5'3' Frame 2 :  AUGCGCCUCCUGCCCCUGCUGGCGCUGCUGGCCCUCUGGGGACCUGACCCAGCCGCAGCCUUUGUGAACCAA
Protein      :  MRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGS

5'3' Frame 3 :  AUGGAAUAA
Protein      :  ME*

3'5' Frame 1 :  AUGCUGGUACAGCAUUGUUCCACAAUGCCACGCUUCUGCAGGGACCCCUCCAGGGCCAAGGGCUGCAGGCUG
Protein      :  MLVQHCSTMPRFCRDPSRAKGCRLPAPGPPPSSTCPTCRSSASRRVLGV*

3'5' Frame 2 :  AUGCCACGCUUCUGCAGGGACCCCUCCAGGGCCAAGGGCUGCAGGCUGCCUGCACCAGGGCCCCCGCCCAGC
Protein      :  MPRFCRDPSRAKGCRLPAPGPPPSSTCPTCRSSASRRVLGV*

3'5' Frame 3 :  AUGGCAGAAGGACAGUGA
Protein      :  MAEGQ*

Generated Protein Sequence : MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAE
Orginal Protein Sequence   : MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAE
Both are same !
```

2. Create a .fasta file with the following content

   >O00626|HUMAN Small inducible cytokine A22.

   MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS

   DS<=

   CPRPGVVLLTFRDKEICADPR

   VPWVKMILNKLSQ

> a. Read the file, extract the header information and print it.

> b. Read and print the sequence from the file.

> c. Append molecular weight of the sequence at the end of the file

Actually should be :

```
>O00626|HUMAN Small inducible cytokine A22.
MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
DSCPRPGVVLLTFRDKEICADPRVPWVKMILNKLSQ
```

```
import re

content = """>O00626|HUMAN Small inducible cytokine A22.
MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
DSCPRPGVVLLTFRDKEICADPRVPWVKMILNKLSQ"""

# content = """>O00626|HUMAN Small inducible cytokine A22.
# MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
# DS<=
# CPRPGVVLLTFRDKEICADPR
# VPWVKMILNKLSQ"""

def rp(text):
    punctuation_pattern = re.compile(r'[^\w\s]')
    return re.sub(punctuation_pattern, '', text)

with open("/content/sequence.fasta", "w") as f:
    f.write(content)

with open("/content/sequence.fasta", "r") as f:
    lines = f.readlines()
    if lines:
        header_info = lines[0].strip()[1:]
        peptide = "".join(line.strip() for line in lines[1:])
        # peptide = rp(peptide)
        print("Header Information:", header_info)
        print("Sequence:", peptide)

def calculate(peptide):
    prot_weight = {
        'A': 89.09, 'R': 174.20, 'N': 132.12, 'D': 133.10, 'C': 121.16,
        'E': 147.13, 'Q': 146.145, 'G': 75.07, 'H': 155.16, 'I': 131.17,
        'L': 131.17, 'K': 146.19, 'M': 149.21, 'F': 165.19, 'P': 115.13,
        'S': 105.09, 'T': 119.12, 'W': 204.23, 'Y': 181.19, 'V': 117.15
    }

    total_weight = sum(prot_weight.get(aa.upper(), 0) for aa in peptide)
    total_weight -= 18.01528 * (len(peptide) - 1)


    return round(total_weight,2)


molecular_weight= calculate(peptide)

with open("/content/sequence.fasta", "a") as f:
    f.write(f"\nMolecular Weight: {molecular_weight}")

with open("/content/sequence.fasta", "r") as f:
    updated_content = f.read()
    print("\nUpdated File Content:")
```

```
print(updated_content)
```

```
Header Information: O00626|HUMAN Small inducible cytokine A22.
Sequence: MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTSDSCPRPGVVLLTFRDKEICAD

Updated File Content:
>O00626|HUMAN Small inducible cytokine A22.
MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
DSCPRPGVVLLTFRDKEICADPRVPWVKMILNKLSQ
Molecular Weight: 10580.45
```

3. Compute the Number of Times a Pattern Appears in a Text

Description:

> This is the first problem in a collection of "code challenges" to accompany $Bioinformatics\ Algorithms: An\ Active-Learning\ Approach$ by Phillip Compeau & Pavel Pevzner.
>
> > A k-mer is a string of length k. We define Count(Text, Pattern) as the number of times that a k-mer Pattern appears as a substring of Text.
> >
> > For example, Count(ACAACTATGCATACTATCGGGAACTATCCT,ACTAT)=3. We note that Count(CGATATATCCATAG, ATA) is equal to 3 (not 2) since we should account for overlapping occurrences of Pattern in Text.

Implement PatternCount

> Given: {DNA strings}} Text and Pattern.
>
> Return: Count(Text, Pattern).

Pseudocode:

PatternCount(Text, Pattern)

> count ← 0
>
> for i ← 0 to |Text| − |Pattern| if Text(i, |Pattern|)
>
> > = Pattern count ← count + 1

> return count

Sample Dataset

> GCGCG

> GCG

Sample Output

> 2

Real Dataset

Input:

> Text : Vibrio Cholerae Oric DataSet

> Pattern: ATGATCAAG

Output:

> 3

Optional : Visit http://rosalind.info/problems/ba1a/ . Solve the problem. Use the sample dataset given in the site.

```
def PatternCount(Text, Pattern):
    count = 0
    pattern_length = len(Pattern)

    for i in range(len(Text) - pattern_length + 1):
        if Text[i:i+pattern_length] == Pattern:
            count += 1

    return count


Text = "GCGCG"
Pattern = "GCG"

result = PatternCount(Text, Pattern)
print(result)
```

```
2
```

```
with open('/content/VibrioCholeraOric.txt', 'r') as f:
    Text = f.read()

Text = Text.upper()
Pattern = "ATGATCAAG"

result = PatternCount(Text, Pattern)
print(result)
```

```
3
```

```
Text = "ATCAATGATCAACGTAAGCTTCTAAGCATGATCAAGGTGCTCACACAGTTTATCCACAACCTGAGTGGATGACATCAAGATAGG
Pattern = "ATGATCAAG"

result = PatternCount(Text, Pattern)
print(result)
```

```
3
```

Optional :

```
Text = "ACAACTATGCATACTATCGGGAACTATCCT"
Pattern = "ACTAT"
result = PatternCount(Text, Pattern)
print(result)
```

```
3
```

```
Text = "CGATATATCCATAG"
Pattern = "ATA"
result = PatternCount(Text, Pattern)
print(result)
```

```
3
```

```
Text = "TGGGACTATACAGTGCTTAGGGACTATTTCCTAATAGGGACTAGGGACTACTCCTCAACAAAACGGGACTAGATGGGGACTAGG
Pattern = "GGGACTAGG"

result = PatternCount(Text, Pattern)
print(result)
```

```
41
```

4. Find All Occurrences of a Pattern in a DNA String

Description:

In this problem, we ask a simple question: how many times can one string occur as a substring of another? Recall from “Find the Most Frequent Words in a String” that different occurrences of a substring can overlap with each other. For example, ATA occurs three times in CGATATATCCATAG.Pattern Matching Problem

Find all occurrences of a pattern in a string.

> Given: Strings Pattern and Genome.

> Return: All starting positions in Genome where Pattern appears as a substring.

> Use 0-based indexing.

Sample Dataset:

> ATAT

> GATATATGCATATACTT

Sample Output:

> 1 3 9

Real Dataset:

> Vibrio Cholerae Genome DataSet

> Pattern: ATGATCAAG

Output:

> 116556 149355 151913 152013 152394 186189 194276 200076 224527
> 307692 479770 610980 653338 679985 768828 878903 985368

Visit http://rosalind.info/problems/ba1d/ . Solve the problem. Use the sample dataset given in the site.

```
def PatternMatching(Pattern, Genome):
    positions = []
    pattern_length = len(Pattern)

    for i in range(len(Genome) - pattern_length + 1):
        if Genome[i:i+pattern_length] == Pattern:
            positions.append(i)

    return positions
```

```
Genome = 'GATATATGCATATACTT'
Pattern = 'ATAT'
result = PatternMatching(Pattern, Genome)
print(" ".join(map(str, result)))
```

```
1 3 9
```

```
with open('/content/VibrioCholeraGenome.txt', 'r') as f:
    Genome = f.read()

Pattern = "ATGATCAAG"

result = PatternMatching(Pattern, Genome)
print(" ".join(map(str, result)))
```

```
116556 149355 151913 152013 152394 186189 194276 200076 224527 307692 479770 610980 6533
```

Optional:

```
Genome = 'CGATATATCCATAG'
Pattern = 'ATA'

result = PatternMatching(Pattern, Genome)
print(" ".join(map(str, result)))
```

```
2 4 10
```

```
Pattern = "TACTCACTA"
Genome = "TACTCACCCGTACTCACTGTACTCACGTACTCACTACTCACTACTCACATACTCACTTACTCACTTACTCACTACTCACGTT
```

```
result = PatternMatching(Pattern, Genome)
print(" ".join(map(str, result)))
```

```
27 34 65 83 90 134 157 178 202 294 311 329 358 370 377 410 417 424 431 461 528 535 550 5
```

5. Find the Most Frequent Words in a String

   Description: We say that Pattern is a **most frequent k-mer** in Text if it maximizes Count(Text, Pattern) among all k-mers. For example, "ACTAT" is a most frequent 5-mer in "ACAACTATGCATCACTATCGGGAACTATCCT", and "ATA" is a most frequent 3-mer of "CGATATATCCATAG".

   Frequent Words Problem

Find the most frequent k-mers in a string.

Given: A DNA string Text and an integer k.

Return: All most frequent k-mers in Text (in any order).

Sample Dataset

> ACGTTGCATGTCGCATGATGCATGAGAGCT
>
> 4

Sample Output

> CATG GCAT

Real Dataset

> Vibrio Cholerae Oric DataSet
>
> K= 9
>
> Output:
>
> > atgatcaag cttgatcat tcttgatca ctcttgatc

Optional: Visit http://rosalind.info/problems/ba1b/ . Solve the problem. Use the sample dataset given in the site

```
def FrequentWords(Text, k):
    kmer_counts = {}
    max_count = 0
    frequent_kmers = []
    for i in range(len(Text) - k + 1):
        kmer = Text[i:i+k]
        kmer_counts[kmer] = kmer_counts.get(kmer, 0) + 1
        max_count = max(max_count, kmer_counts[kmer])
    for kmer, count in kmer_counts.items():
        if count == max_count:
            frequent_kmers.append(kmer)

    return frequent_kmers


Text = 'ACGTTGCATGTCGCATGATGCATGAGAGCT'
k = 4
result = FrequentWords(Text, k)
print(" ".join(sorted(result)))
```

```
        CATG GCAT

    with open('/content/VibrioCholeraOric.txt', 'r') as f:
```