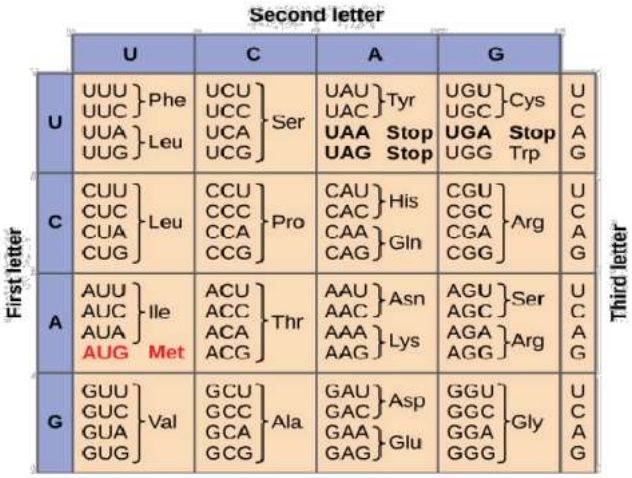
22BIO201 Intelligence of Biological Systems 1

Lab Sheet 2

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.




```
codon_dict = {
    'A': ['GCU', 'GCC', 'GCA', 'GCG'],
    'C': ['UGU', 'UGC'],
    'D': ['GAU', 'GAC'],
    'E': ['GAA', 'GAG'],
    'F': ['UUU', 'UUC'],
    'G': ['GGU', 'GGC', 'GGA', 'GGG'],
    'H': ['CAU', 'CAC'],
    'I': ['AUU', 'AUC', 'AUA'],
    'K': ['AAA', 'AAG'],
    'L': ['UUA', 'UUG', 'CUU', 'CUC', 'CUA', 'CUG'],
    'M': ['AUG'],
    'N': ['AAU', 'AAC'],
    'P': ['CCU', 'CCC', 'CCA', 'CCG'],
    'Q': ['CAA', 'CAG'],
    'R': ['CGU', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'],
    'S': ['UCU', 'UCC', 'UCA', 'UCG', 'AGU', 'AGC'],
    'T': ['ACU', 'ACC', 'ACA', 'ACG'],
    'V': ['GUU', 'GUC', 'GUA', 'GUG'],
    'W': ['UGG'],
    'Y': ['UAU', 'UAC'],
    '_': ['UAA', 'UAG', 'UGA']  # Stop codons
}

"""
Each Amino acid has been mapped to the corresponding codon patterns
Total 21 amino acids has been mapped
Each Amino acid has multiple different RNA codon pattern
"""
```

```
'\nEach Amino acid has been mapped to the corresponding codon patterns\nTotal 21 amino
acids has been mapped\nEach Amino acid has multiple different RNA codon pattern\n"
```

```
dna = "AGCCCTCCAGGACAGGCTGCATCAGAAGAGGCCATCAAGCAGGTCTGTTCCAAGGGCCTTTGCGTCAGGTGGGCTCAGGATTCCAGGGTGGCTGGACCCCAGGCCCCAGCTCTGCAGCAGGGAGGACGTGGCTG
rna = dna.replace('T', 'U')

"""
```

```
The Test DNA strand has been converted into RNA
The function rna2protein() takes in the rna and codon table and converts it into the protein chain or peptide chain
"""

def rna2protein(rna, codon_table):
  protein_sequence = ""
  rna = [rna[i:i+3] for i in range(0, len(rna), 3)]
  for protein in rna:
    for acid, codons in codon_table.items():
      if protein in codons:
        protein_sequence +=acid
        break
  return protein_sequence


protein = rna2protein(rna, codon_dict)

print("RNA Sequence:")
print(rna)

print("\n\nAmino Acid Sequence:")
print(protein)
```

```
CAUGUGGGGGUGAGCCCAGGGGCCCCAAGGCAGGGCACCUGGCCUUCAGCCUGCCUCAGCCCUGCCUGUCUCCCAGAUCACUGUCCUUCUGCCAUGGCCCUGUGGAUGCGCCUCCUGCCCCUGCUGGCGCUGCUGGC


GAGRGPWCRQPAALGPGGVPAEAWHCGTMLYQHLLPLPAGELLQLDAARRQPHTRRLLHRERWNKALEP
```

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.

```
content = """>000626 | HUMAN Small inducible cytokine A22.
MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
DSCPRPGVVLLTFRDKEICADPR
VPWVKMILNKLSQ
"""

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


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

with open("sequence.fasta", "r") as f:
  print(f"Header: {f.readline()}")
```

```
Header: >000626 | HUMAN Small inducible cytokine A22.
```

```
#b. Read and print the sequence from the file.

with open("sequence.fasta", "r") as f:
  f.readline()
  while True:
    line = f.readline()
    if not line:
      break
    print(line.strip())
```

```
MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
DSCPRPGVVLLTFRDKEICADPR
VPWVKMILNKLSQ
```

```
#c. Append molecular weight of the sequence at the end of the file.

acid_weights = {
    'A': 89.093, 'C': 121.158, 'D': 133.103, 'E': 147.129,
    'F': 165.192, 'G': 75.067, 'H': 155.155, 'I': 131.175,
    'K': 146.189, 'L': 131.175, 'M': 149.208, 'N': 132.118,
    'P': 115.132, 'Q': 146.146, 'R': 174.203, 'S': 105.093,
    'T': 119.119, 'V': 117.146, 'W': 204.228, 'Y': 181.191
}

"""
The Molecular weights of the Amino Acids has been taken from online references
First the Peptide sequence is read from the file
Then for each Amino Acid, The Molecular Weight is referenced from The dictionary
And summed up to find the Total Molecular Weight of the peptide Chain
The Molecular Weight is then attached to the End of the File
"""

seq=""
with open("sequence.fasta", "r") as f:
  f.readline()
  while True:
    line = f.readline()
    if not line:
      break
    seq+=line.strip()

print(f"Amino Acid Sequence: {seq}")
mweight=0
for i in seq:
  mweight+=acid_weights[i]

print(f"\nMolecular Weight of the Amino Acid Sequence: {round(mweight, 2)}u.")

with open("sequence.fasta", "a") as f:
  f.write(f"\nMolecular Weight of the Amino Acid Sequence: {round(mweight, 2)}u.")
```

```
Amino Acid Sequence: MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTSDSCPRPGVVLLTFRDKEICADPRVPWVKMILNKLSQ

Molecular Weight of the Amino Acid Sequence: 12237.93u.
```

```
with open("sequence.fasta", "r") as f:
  print(f.read())
```

```
>000626 | HUMAN Small inducible cytokine A22.
MARLQTALLVVLVLLAVALQATEAGPYGANMEDSVCCRDYVRYRLPLRVVKHFYWTS
DSCPRPGVVLLTFRDKEICADPR
VPWVKMILNKLSQ

Molecular Weight of the Amino Acid Sequence: 12237.93u.
```

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*).

```
"""
The count(dna, pattern) function takes in the DNA strand and the K-mer to be found
The Function slides the pattern on top of the DNA strands
And counts the Number of Occurrence of the Pattern in The DNA strand
"""

def count(dna, pattern):
  k=len(pattern)
  cnt=0
  for i in range(len(dna)-k):
    if dna[i:i+k] == pattern:
      cnt+=1
  return cnt

def count2(dna, pattern): # This Function does the same stuff as count() but more efficiently
    return [dna[i:i+len(pattern)] for i in range(len(dna)-len(pattern))].count(pattern)


print(count("ACAACTATGCATACTATCGGGAACTATCCT", "ACTAT"))
print(count2("ACAACTATGCATACTATCGGGAACTATCCT", "ACTAT"))
```

```
3
3
```

```
oric = """atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttttacggaagaatgatcaagctgctgctcttgatcatcgtttc"""

oric= oric.upper().replace("\n", "")    # Pre Proccessing ORIC
print(count(oric, "ATGATCAAG"))
```

```
3
```

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

```
def Occurrences(dna, pattern):
  k=len(pattern)
  occurrence=[]
  for i in range(len(dna)-k):
    if dna[i:i+k] == pattern:
      occurrence.append(i)
  return occurrence

def Occurrences2(dna, pattern):
    return [i for i in range(len(dna)-len(pattern)) if dna[i:i+len(pattern)] == pattern]


from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving VibrioCholeraGenome.txt to VibrioCholeraGenome.txt

```
with open("VibrioCholeraGenome.txt", 'r') as f:
  genome = f.read()
  print(genome)


print(Occurrences(genome, "ATGATCAAG"))
print(Occurrences2(genome, "ATGATCAAG"))
```

```
[116556, 149355, 151913, 152013, 152394, 186189, 194276, 200076, 224527, 307692, 479770, 610980, 653338, 679985, 768828, 878903, 985368]
[116556, 149355, 151913, 152013, 152394, 186189, 194276, 200076, 224527, 307692, 479770, 610980, 653338, 679985, 768828, 878903, 985368]
```

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.

```
def frequentwords(genome, k):
  kmers = list(set([genome[i:i+k] for i in range(len(genome)-k)]))
  freq={}
  for i in kmers:
    freq[i] = count(genome, i)
  freq = dict(sorted(freq.items(), key=lambda item: item[1], reverse=True))
  return freq

def frequentwords2(genome, k): # This is also a compacted Function of frequentwords() which is more efficient using List and Dictionary compr
  return {kmer: genome.count(kmer) for kmer in sorted(set([genome[i:i+k] for i in range(len(genome)-k)]), key=lambda x: genome.count(x), reve

print(frequentwords("ACGTTGCATGTCGCATGATGCATGAGAGCT", 4))
print(frequentwords2("ACGTTGCATGTCGCATGATGCATGAGAGCT", 4))
```

```
{'GCAT': 3, 'CATG': 3, 'ATGA': 2, 'TGCA': 2, 'GAGA': 1, 'TGAT': 1, 'ACGT': 1, 'TTGC': 1, 'GTTG': 1, 'CGTT': 1, 'TGTC': 1, 'GATG': 1, 'ATC
{'GCAT': 3, 'CATG': 3, 'ATGA': 2, 'TGCA': 2, 'GAGA': 1, 'TGAT': 1, 'ACGT': 1, 'TTGC': 1, 'GTTG': 1, 'CGTT': 1, 'TGTC': 1, 'GATG': 1, 'ATC
```