**Name : D.P.Bhagya Hendalage**
**Registration number : 2017s16415**
**Index number : s13674**

# BT-3172: Special Topics in Bioinformatics: Practical Computing for Biologists
# Final Project Assignment

Project 1: Sequence analyzer
The Python program should include classes and methods to analyze biological sequences (DNA, RNA and amino acid). More specifically, it should include the following methods. Please note that you cannot use Biopython packages for this program.

## A method to return the sequence type.

```
def get_Seq_Type(sequence):
    """
            A method to check the sequence type from all 3 sequence types:
DNA, mRNA, amino acid.

            Input: sequence
            Output: sequence type ('DNA' or 'mRNA' or 'Amino acid')
            """
    if 'M' in sequence:
        return "Amino acid"
    elif 'U' in sequence and 'M' not in sequence:
        return "mRNA"
    else:
        return "DNA"
```

## Methods to calculate nucleotide base counts in DNA and RNA sequences. The same methods should be able to plot a graph to compare each base count and save it in a figure.

DNA

```
def get_Character_Count(self):
    """
            method for return all characters in DNA sequence and its count as
a dictionary
            Input: sequence
            Output: dictionary of character counts,different bases compared
using diagram
    """
    character_count = {}

    for i in self.sequence:
        if i in character_count:
            character_count[i] += 1
        else:
            character_count[i] = 1
```

```
    #to get a graph
    plt.bar(character_count.keys(), character_count.values())
    plt.title("Graph of each base in the sequence and its' frequencies")
    plt.xlabel("different bases")
    plt.ylabel("Counts")
    plt.savefig('DNA_character.png')
    plt.show()
    return (character_count)
```

RNA

```
def get_Character_Count(self):
    """
                    method for return all characters in mRNA sequence and its
count as a dictionary
                    Input: sequence
                    Output: dictionary of character counts,different bases
compared using diagram
            """
    character_count = {}

    for i in self.sequence:
        if i in character_count:
            character_count[i] += 1
        else:
            character_count[i] = 1
    # to get graph
    plt.bar(character_count.keys(), character_count.values())
    plt.title("Graph of each base in the sequence and its' frequencies")
    plt.xlabel("different bases")
    plt.ylabel("Counts")
    plt.savefig('mrna_character.png')
    plt.show()
    return character_count
```

## Methods to calculate AT content in DNA and RNA sequences.

DNA

```
def get_AT_Contents(self):
    '''
            method for get A,T bases content in sequence
            Input : sequence
            Output : AT content
    '''
    sequence = self.sequence
    AT_content = (sequence.count("A") + sequence.count("T")) / len(sequence)
    return AT_content
```

RNA

```
def get_AT_Contents(self):
    '''
                    method for get A,U bases content in sequence
                    Input : sequence
```

```
                    Output : AT content
    '''
    sequence = self.sequence
    AT_content = (sequence.count("A") + sequence.count("U")) / len(sequence)
    return AT_content
```

## Methods to calculate GC content in DNA and RNA sequences.

DNA

```
def get_GC_Contents(self):
    '''
                    method for get G,C bases content in sequence
                    Input : sequence
                    Output : AT content
    '''
    sequence = self.sequence
    GC_content = (sequence.count("G") + sequence.count("C")) / len(sequence)
    return GC_content
```

RNA

```
def get_GC_Contents(self):
    '''
                    method for get G,C bases content in sequence
                    Input : sequence
                    Output : GC content
    '''
    sequence = self.sequence
    GC_content = (sequence.count("G") + sequence.count("C")) / len(sequence)
    return GC_content
```

## A method to compare AT and GC contents of two given input nucleotide sequences using a suitable graph.

```
def compare_AT_GC(self):
    '''
            method for get graph of list of AT and GC contents of two given
nucleic acid sequence
            input : dictionary of AT and GC content of two sequences
            output : graph of AT and GC contents of two sequences
    '''
    df = pd.DataFrame(seq)
    df.index = ['AT', 'GC']
    print(df)
    # plot grouped bar chart
    df.plot(kind="bar")
    plt.savefig("Compare_AT_GC_two_seq.png")
    plt.show()
```

## A method to transcribe an DNA sequence.

```
def transcribe_Sequence(self):
    '''
```

```
            method for transcribe DNA sequence
            Input : sequence
            Output : mRNA
    '''
    transcribed_sequence=self.sequence.replace('T','U')
    return transcribed_sequence
```

## A method to generate all possible combinations of 3-mers and 4-mers given the constituent bases.

```
def get_3mers_4mers(self):
    '''
            method for find all possible 3-mers and 4-mers in a sequence
            input : sequence
            output : 3-mers and 4 mers as a list
    '''
    seq = self.sequence
    three_mers = [seq[i:i + 3] for i in range(len(seq) - 2)]
    four_mers = [seq[i:i + 4] for i in range(len(seq) - 3)]
    return three_mers,four_mers
```

## A method to translate an mRNA sequence.

```
def upload_Codons(self):
    '''
                    method for get codon and corresponding amino acid as a
dictionary
                    Input : codon_table.txt
                    Output : Codon_dict
    '''
    Codon_dict = {}
    # open codon_table and store as a variable
    with open("codon_table.txt", 'r') as codon_table:
        for line in codon_table:
            # remove header and empty lines
            if '#' not in line and line != '\n':
                (codon, AminoAcid, Letter, FullName) =
line.strip().split('\t')
                # fill the dictionary using codons and related amino acid
name
                Codon_dict[codon] = Letter
    return(Codon_dict)

def translate_Sequence(self):
    '''
                            method for get translate mRNA sequence to protein
                            Input : sequence, codon dictionary
                            Output : protein
    '''
    protein = ''
    Codon_map = self.upload_Codons()
    #start when meet 'AUG'
    sequence=self.sequence
    start = sequence.find('AUG')
```

```
    if start != -1:
        while start + 2 < len(sequence):
            codon = sequence[start:start + 3]
            #stop when meet stop codons
            if codon == "UAG" or codon == "UAA" or codon == "UGA":
                break;
            start += 3
            for i in Codon_map[codon]:
                protein += i
        return protein
```

## A method to calculate the hydrophobicity of a given amino acid sequence.

```
def get_Hydrophobicity(self):
    '''
            method for find hydrophobicity using content of hydrophobic amino
acids
            input : sequence
            output : hydrophobicity
    '''
    # hydrophobic amino acids as a list
    HY = ['A', 'I', 'L', 'F', 'M', 'W', 'Y', 'V']
    count = 0
    for i in self.sequence:
        if i in HY:
            count += 1
    Hydrophobicity = (count / len(sequence))
    return (Hydrophobicity)
```

## A method to calculate the molecular weight of a given amino acid sequence.

```
def get_molecular_weight(self):
    '''
            method for find molecular weight using all amino acids in a
protein and their moleecular weights
            input : sequence
            output : molecular weight
    '''
    # all amino acids in a peptide chain and their molecular weight as a
dictionary
    MW = {'A': 89.1, 'R': 174.2, 'N': 132.1, 'D': 133.1, 'C': 121.2, 'E':
147.1, 'Q': 146.2, 'G': 75.1, 'H': 155.2,
          'I': 131.2, 'L': 131.2, 'K': 146.2, 'M': 149.2, 'F': 165.2, 'P':
115.1, 'S': 105.1, 'T': 119.1,
          'W': 204.2, 'Y': 181.2, 'V': 117.1}
    mwscore = 0
    for aa in self.sequence:
        mwval = MW[aa]
        mwscore += mwval
    return mwscore
```

Full code:

```
'''
Author : Bhagya Hendalage
Date : 06/02/2021

Python methods to analyze DNA sequences, RNA sequences , amino acid sequences

Input: DNA/mRNA/amino acid sequences in FASTA format
Output: Sequence class,DNAseq subclass,mRNAseq subclass,AAseq subclass,
Nucleotide subclass
'''
# import packages
import matplotlib.pyplot as plt
import pandas as pd
#create a Sequence class
class Sequence:
    # define attributes and methods
    def __init__(self, accession_number, name, sequence):
        self.sequence = sequence
        self.accession_number = accession_number
        self.name = name
        self.length = len(sequence)

    def fasta_Split(file_name):
        """
                method to split fasta file to dictionary

                Input: fasta file_name
                Output: dictionary with keys and values
                """
        # define an empty dictionary
        dict = {}
        with open(file_name, 'r') as sequence:
                gene_name = ''
                seq = ''
                for line in sequence:
                    if line != '\n':
                        line = line.strip()
                        if '>' in line:
                            # to split from ','
                            full_header = line.split(',')
                            # to take the first index value as header
                            header = full_header[0]
                            header_parts = header.split(' ')
                            # to take all indeces to value
                            gene_name = header_parts[0:]
                            accession_num = header_parts[0]
                            # without seq='', all keys contain first iterated
sequence
                            seq = ''
                        if '>' not in line:
                            seq = seq + line
                            dict[accession_num] = gene_name, seq
        return (dict)

    def get_Seq_Type(sequence):
        """
                A method to check the sequence type from all 3 sequence
```

```
types: DNA, mRNA, amino acid.

                Input: sequence
                Output: sequence type ('DNA' or 'mRNA' or 'Amino acid')
                """
        if 'M' in sequence:
            return "Amino acid"
        elif 'U' in sequence and 'M' not in sequence:
            return "mRNA"
        else:
            return "DNA"

#subclass for DNA
class DNAseq(Sequence):
    def __init__(self, accession_number, name, sequence):
        super().__init__(accession_number, name, sequence)

    def get_Character_Count(self):
        """
                method for return all characters in DNA sequence and its
count as a dictionary
                Input: sequence
                Output: dictionary of character counts,different bases
compared using diagram
        """
        character_count = {}

        for i in self.sequence:
            if i in character_count:
                character_count[i] += 1
            else:
                character_count[i] = 1

        #to get a graph
        plt.bar(character_count.keys(), character_count.values())
        plt.title("Graph of each base in the sequence and its' frequencies")
        plt.xlabel("different bases")
        plt.ylabel("Counts")
        plt.savefig('DNA_character.png')
        plt.show()
        return (character_count)

    def get_AT_Contents(self):
        '''
                method for get A,T bases content in sequence
                Input : sequence
                Output : AT content
        '''
        sequence = self.sequence
        AT_content = (sequence.count("A") + sequence.count("T")) /
len(sequence)
        return AT_content

    def get_GC_Contents(self):
        '''
                        method for get G,C bases content in sequence
                        Input : sequence
```

```
                    Output : AT content
        '''
        sequence = self.sequence
        GC_content = (sequence.count("G") + sequence.count("C")) /
len(sequence)
        return GC_content
    def transcribe_Sequence(self):
        '''
                method for transcribe DNA sequence
                Input : sequence
                Output : mRNA
        '''
        transcribed_sequence=self.sequence.replace('T','U')
        return transcribed_sequence


# A subclass for mRNA sequences
class mRNAseq(Sequence):
    def __init__(self, accession_number, name, sequence):
        super().__init__(accession_number, name, sequence)

    def get_Character_Count(self):
        """
                    method for return all characters in mRNA sequence and
its count as a dictionary
                    Input: sequence
                    Output: dictionary of character counts,different bases
compared using diagram
                """
        character_count = {}

        for i in self.sequence:
            if i in character_count:
                character_count[i] += 1
            else:
                character_count[i] = 1
        # to get graph
        plt.bar(character_count.keys(), character_count.values())
        plt.title("Graph of each base in the sequence and its' frequencies")
        plt.xlabel("different bases")
        plt.ylabel("Counts")
        plt.savefig('mrna_character.png')
        plt.show()
        return character_count

    def get_AT_Contents(self):
        '''
                    method for get A,U bases content in sequence
                    Input : sequence
                    Output : AT content
        '''
        sequence = self.sequence
        AT_content = (sequence.count("A") + sequence.count("U")) /
len(sequence)
        return AT_content

    def get_GC_Contents(self):
```

```
        '''
                        method for get G,C bases content in sequence
                        Input : sequence
                        Output : GC content
        '''
        sequence = self.sequence
        GC_content = (sequence.count("G") + sequence.count("C")) /
len(sequence)
        return GC_content


    def upload_Codons(self):
        '''
                        method for get codon and corresponding amino acid as
a dictionary
                        Input : codon_table.txt
                        Output : Codon_dict
        '''
        Codon_dict = {}
        # open codon_table and store as a variable
        with open("codon_table.txt", 'r') as codon_table:
            for line in codon_table:
                # remove header and empty lines
                if '#' not in line and line != '\n':
                    (codon, AminoAcid, Letter, FullName) =
line.strip().split('\t')
                    # fill the dictionary using codons and related amino acid
name
                    Codon_dict[codon] = Letter
        return(Codon_dict)

    def translate_Sequence(self):
        '''
                                method for get translate mRNA sequence to
protein
                                Input : sequence, codon dictionary
                                Output : protein
        '''
        protein = ''
        Codon_map = self.upload_Codons()
        #start when meet 'AUG'
        sequence=self.sequence
        start = sequence.find('AUG')
        if start != -1:
            while start + 2 < len(sequence):
                codon = sequence[start:start + 3]
                #stop when meet stop codons
                if codon == "UAG" or codon == "UAA" or codon == "UGA":
                    break;
                start += 3
                for i in Codon_map[codon]:
                    protein += i
            return protein


#subclass for amino acids
class AAseq(Sequence):
```

```
    def __init__(self, accession_number, name, sequence, uniprot_ID):
        super().__init__(accession_number, name, sequence)
        self.uniprot_ID = uniprot_ID

    def get_Hydrophobicity(self):
        '''
                method for find hydrophobicity using content of hydrophobic
amino acids
                input : sequence
                output : hydrophobicity
        '''
        # hydrophobic amino acids as a list
        HY = ['A', 'I', 'L', 'F', 'M', 'W', 'Y', 'V']
        count = 0
        for i in self.sequence:
            if i in HY:
                count += 1
        Hydrophobicity = (count / len(sequence))
        return (Hydrophobicity)

    def get_molecular_weight(self):
        '''
                method for find molecular weight using all amino acids in a
protein and their moleecular weights
                input : sequence
                output : molecular weight
        '''
        # all amino acids in a peptide chain and their molecular weight as a
dictionary
        MW = {'A': 89.1, 'R': 174.2, 'N': 132.1, 'D': 133.1, 'C': 121.2, 'E':
147.1, 'Q': 146.2, 'G': 75.1, 'H': 155.2,
              'I': 131.2, 'L': 131.2, 'K': 146.2, 'M': 149.2, 'F': 165.2,
'P': 115.1, 'S': 105.1, 'T': 119.1,
              'W': 204.2, 'Y': 181.2, 'V': 117.1}
        mwscore = 0
        for aa in self.sequence:
            mwval = MW[aa]
            mwscore += mwval
        return mwscore

#subclass for Nucleotide
class Nucleotide(Sequence):
    def __init__(self, accession_number, name, sequence):
        super().__init__(accession_number, name, sequence)

    def get_3mers_4mers(self):
        '''
                method for find all possible 3-mers and 4-mers in a sequence
                input : sequence
                output : 3-mers and 4 mers as a list
        '''
        seq = self.sequence
        three_mers = [seq[i:i + 3] for i in range(len(seq) - 2)]
        four_mers = [seq[i:i + 4] for i in range(len(seq) - 3)]
        return three_mers,four_mers

    def compare_AT_GC(self):
```

```
        '''
                method for get graph of list of AT and GC contents of two
given nucleic acid sequence
                input : dictionary of AT and GC content of two sequences
                output : graph of AT and GC contents of two sequences
        '''
        df = pd.DataFrame(seq)
        df.index = ['AT', 'GC']
        print(df)
        # plot grouped bar chart
        df.plot(kind="bar")
        plt.savefig("Compare_AT_GC_two_seq.png")
        plt.show()

# Implementation
if __name__ == '__main__':
    sequence = Sequence.fasta_Split("sequence (3).fasta")
    print(sequence)
    seq_objects = []

for key, value in sequence.items():
     if Sequence.get_Seq_Type(value[1]) == "DNA":
             seq_objects.append(DNAseq(key, value[0][4], value[1]))
     elif Sequence.get_Seq_Type(value[1]) == "mRNA":
             seq_objects.append(mRNAseq(key, value[0][4], value[1]))
     else:
            seq_objects.append(AAseq(key, value[0][5], value[1],
value[0][6]))
# create objects for four sequences
for objects in seq_objects:
       if objects.accession_number=='>AH002877.2':
         print('i')
         print("GC_Content: ", objects.get_GC_Contents())
         print("AT_Content: ", objects.get_AT_Contents())
         print("charater count:" , objects.get_Character_Count())
         print('transcribed_sequence:',objects.transcribe_Sequence())

       if objects.accession_number=='>DQ659148.1':
         print('ii')
         print("GC_Content: ", objects.get_GC_Contents())
         print("AT_Content: ", objects.get_AT_Contents())
         print("charater count:" , objects.get_Character_Count())
         print('transcribed_sequence:',objects.transcribe_Sequence())

       if objects.accession_number =='>NM_002115.3':
         print('iii')
         new_seq = objects.transcribe_Sequence()
         mRNA = mRNAseq(objects.accession_number,objects.name,new_seq)
         print("AT content: ", mRNA.get_AT_Contents())
         print("GC content: ", mRNA.get_GC_Contents())
         print("Character count: ", mRNA.get_Character_Count())
         print('iv')
         prot = mRNA.upload_Codons()
         print(prot)
         protein = mRNA.translate_Sequence()
         print("protein: ", protein)
         print("length of protein: ", len(prot))
```

```
       if objects.accession_number=='>AAA60129.1':
         print('v')
         print("hydrophobicity of protein: ", objects.get_Hydrophobicity())
         print("Molecular weight of protein: ",
objects.get_molecular_weight())



# create objects for user input sequences
seq_objects_two = []
print(sequence)
for key, value in sequence.items():
     if  Sequence.get_Seq_Type(value[1]) !='Amino acid':
            seq_objects_two.append(DNAseq(key,value[0][4], value[1]))

dict_two = Sequence.fasta_Split("sequence (3).fasta")
# to get the user input
d1 = str(input("Enter seq1 accesion number with > mark: "))
for objects in seq_objects_two:
    if d1==objects.accession_number:
       seq1 = (dict_two[d1][1])
       AT=objects.get_AT_Contents()
       GC=objects.get_GC_Contents()
       # create a list
       seq1=[AT,GC]

# to get the user input
d2 = str(input("Enter seq2 accesion number with > mark: "))
for objects in seq_objects_two:
    if d2==objects.accession_number:
       seq2 = (dict_two[d2][1])
       AT=objects.get_AT_Contents()
       GC=objects.get_GC_Contents()
       # create a list
       seq2=[AT,GC]
# create a dictionary from two lists
seq={}
for i in('seq1','seq2'):
    seq[i]=locals()[i]
# input created dictionary to the "compare_AT_GC" method
Nucleotide.compare_AT_GC(seq)

# to create objects for all nucleotide sequences to get 3_mers and 4-mers
seq_objects3mers_4mers=[]
for key, value in sequence.items():
     if Sequence.get_Seq_Type(value[1]) != "Amino acid":
            seq_objects3mers_4mers.append(Nucleotide(key, value[0][4],
value[1]))
for objects in seq_objects3mers_4mers:
    if objects.accession_number == '>AH002877.2':
       print("seq1 3mers_4-mers:",objects.get_3mers_4mers())
    if objects.accession_number == '>DQ659148.1':
       print(">seq2 3mers_4-mers:",objects.get_3mers_4mers())
    if objects.accession_number == '>NM_002115.3':
       print(">seq3 3mers_4-mers:",objects.get_3mers_4mers())
```