BT-3172: Special Topics in Bioinformatics: Computing for Biologists
Lab 3: Use of Python functions and object-oriented programming concepts in bioinformatics.

W.M. Ayesha Sanahari | S13722 | 2017s16470

In this practical you will learn how to write custom methods and use Object-oriented programming (OOP) concepts in Python to solve biological problems.

For this practical, you will be working with several genes and proteins involved with the DREB pathway. This is an ABA-independent pathway, which is important in plant abiotic stress response.

After using PyCharm to write your scripts, copy the codes to the appropriate space below the questions. Also, submit the Python files separately so they can be tested. Use the following format to name the script: YourIndexNo_PrimaryQuestionNo_SecondaryQuestionNo.py.

1) Writing custom Python methods to analyze DNA sequences *(25 marks)*

   I. Use the UniProt knowledgebase to search for the following proteins in rice japonica subspecies: DREB1A, DREB1B, DREB2A, and DREB2B.isoform 1. Write their UniProt identifier numbers in front of the protein names below. Mention their reviewed status in front of the ID. Access their amino acid sequences in FASTA format. Obtain the RefSeq gene entry record for each protein and write their RefSeq gene IDs in front of the correct record. Then locate their mRNA sequences and access **only the coding sequence** for each entry in FASTA format. Create an empty FASTA file named "OSDREB_sequences.FASTA" and copy the above amino acid sequences one after another. Use the following header format to name the FASTA header for each entry.
   For proteins: >Gene_name**_P**- RefSeq_gene_ID-species-subspecies-Uniprot_ID-reviewed_status
   For coding sequences: >Gene_name**_CDS**-RefSeq_gene_ID-species-subspecies *(6 marks).*

	UniProt identifier numbers	Reviewed Status	RefSeq gene ID
DREB1A	Q64MA1	Reviewed	4347620
DREB1B	Q3T5N4	Reviewed	4347618
DREB2A	Q0JQF7	Reviewed	4324418
DREB2B	Q5W6R4	Reviewed	4338484

   II. Write the equation and an algorithm to calculate the AT content of a given DNA or mRNA sequence. *(4 marks)*

AT_content = (A_count + T_count) / length

Read the sequence
Define counter for A = 0
Define counter for T = 0
For each letter in sequence

```
        If 'A' is found
                A_counter += 1
        If 'T' is found
                T_counter += 1
    Calculate sequence length
    AT_content = (A_count + T_count) / sequence length
    Print (AT_content)
```

III. Implement the above algorithm as a custom method in Python. For the sub questions: III, IV, V and VI, use a single Python script to write the 3 methods and the implementation. You can save it only using the main question number. *(5 marks)*

IV. Write a custom Python method to split multiple FASTA sequences in a single text file and return a dictionary containing sequence headers as keys and the sequences as values. *(3 marks)*

V. Write a custom Python method to check the type of a given sequence is DNA, mRNA or amino acid sequence. *(3 marks)*

VI. Use the above written methods to check each sequence in the OSDREB_sequences.FASTA file and print the AT content. *(4 marks)*

```
"""
Author : Ayesha Sanahari
date : 06/Dec/2020
Writing custom Python methods to analyze DNA sequences

Calculating the AT content of a given DNA or mRNA sequence
Input: DNA/mRNA sequences in FASTA format
Output: AT content of the given DNA or mRNA sequence
"""
class Sequence:

    def get_AT_content(dna):
        length = len(dna)
        A_count = dna.upper().count('A')
        T_count = dna.upper().count('T')
        AT_content = (A_count + T_count) / length
        return round(AT_content, 3)

    @staticmethod
    def splitingFastaFile(file_name):

        # Define an empty dictionary to store the fasta sequences
        seq_dict = {}
```

```
        # open/read fasta file (with many fasta records) & store it in a variable
        with open(file_name,'r') as sequenceFile:
            for line in sequenceFile:
                # Remove blank lines
                if line != '\n':
                    line = line.strip()

                    # seperate the headers with '>' as the keys in the dictionary
                    if ('>') in line:
                        header = line
                        seq_dict[header]=''
                    # if line not begin with '>', take the sequences as the values of dictionary
                    else:
                        seq_dict[header] += line.strip()
        return seq_dict

    def getType(sequence):
        if 'M' in sequence:
            return "Amino acid"
        elif 'U' in sequence:
            return "mRNA"
        else:
            return "DNA"


dic = Sequence.splitingFastaFile("OSDREB_sequences.fasta")
print(dic)

for key,value in dic.items():
    print(key)
    print('Type : ',Sequence.getType(value))

    if 'M' not in value:
        print('AT_content : ', Sequence.get_AT_content(value),'\n')
    else:
        print('AT_content : Not Found \n')
```

2) Familiarizing with Python OOP techniques. Writing a Sequence class. Use the same Python script to write the class and subclasses for sub questions I, II, III and IV. (75 marks)

I. Write a Python Sequence class to store any biological sequence (DNA, mRNA, and amino acid sequences). It should have the following attributes and methods: (20 marks)

Attributes

i. Gene ID.

ii. Gene name.
iii. Sequence type
iv. Sequence length.
v. Sequence count (to count the number of sequences created by the sequence class).
vi. Species name.
vii. Subspecies name

Methods

viii. Constructer method to create sequence objects
ix. fasta_Split(): a static method to split multiple FASTA sequences in a single text file and return a dictionary containing the Gene name (first item in the hyphen-separated list) as the key and a list containing hyphen-separated fields in the header plus the sequence as the value. Make sure the Gene name (key) is also included in the value list (refer to 1.IV and modify the code accordingly).
x. get_Seq_Type() A method to check the sequence type (refer to 1V), but this time, distinguish between all 3 sequence types: DNA, mRNA, amino acid.
xi. get_Character_Count(): this should return a dictionary of character counts with each character as the key and count as the value. A character can be a nucleobase or an amino acid.

```
"""
Author : Ayesha Sanahari
date : 20/Dec/2020
Familiarizing with Python Object-oriented programming (OOP) techniques.
Writing custom Python methods to analyze DNA sequences

Eg: Calculating the AT content of a given DNA or mRNA sequence
    Input: DNA/mRNA sequences in FASTA format
    Output: AT content of the given DNA or mRNA sequence
"""
class Sequence:
    sequence_count = 0

    def __init__(self, sequence, Gene_name, Gene_ID, Species_name, Subsp_name):
        self.Gene_ID = Gene_ID
        self.Gene_name = Gene_name
        self.Seq_Type = Sequence.get_Seq_Type(sequence)
        self.Seq_length = len(sequence)
        self.Species_name = Species_name
        self.Subsp_name = Subsp_name
        Sequence.sequence_count += 1

    @staticmethod
    def fasta_Split(file_name):
        """
        A static method to split multiple FASTA sequences in a single text file
        and return a dictionary containing the Gene name
        (first item in the hyphen-separated list) as the key and
        a list containing hyphen-separated fields in the header plus the sequence as the value.
```

```
    Input: Argument - file_name
    Output: dictionary containing keys and values
    """

    seq_dict = {}
    with open(file_name, 'r') as sequenceFile:

        for line in sequenceFile:
            # Remove blank lines
            if line != '\n':
                line = line.strip()

                # seperate the headers with '>' as the keys in the dictionary
                if ('>' in line) :
                    #the fasta header includes the ">" sign too, so to remove it
                    key_list = line.strip('>').split('-')
                    key = key_list[0]
                    seq_dict[key] = []
                    sequence = ''

                else:
                    sequence += line
                    seq_dict[key] = [sequence]

                    for i in key_list:
                        # seq  dict[key].insert(0,i)
                        seq_dict[key].append(i)
    return (seq_dict)

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

    Input: Argument - sequence
    Output: sequence type (DNA or mRNA or amino acid)
    """

    amino_acid = ['M','N','I','R','K','Q','E','S','P','L','O','H','T','V','W','D','Y']

    if 'U' in sequence:
        return 'mRNA'
    else:
        for letter in sequence:
            if letter in amino_acid:
                return 'Amino acid'
            else:
                return 'DNA'
```

```
def get_Character_Count(sequence):
    """
    this method should return a dictionary of character counts with
    each character as the key and count as the value.
    A character can be a nucleobase or an amino acid.

    Input: Argument - sequence
    Output: dictionary of character counts
    """
    character_dic ={}            # initialize a dictionary

    for ch in sequence:          #read through each character in sequence
        if ch in character_dic:
            character_dic[ch] += 1   #if it's been seen before, increment counter
        else:
            character_dic[ch] = 1    #otherwise, insert it into the dictionary

    return character_dic
```

II. Write a subclass of the Sequence class for DNA sequences named "DNAseq". It should have the following unique/additional attributes and methods. (5 marks)

Attributes

i. AT_content.
ii. Transcribed sequence

Methods

iii. Constructer method to create DNA sequence objects
iv. transcribe_Sequence(): transcribe the given DNA sequence into its mRNA sequence and store it in the Transcribed sequence instance variable.
v. get_ATcontent(): this should return the AT content of the given sequence and update the AT_content instance variable.

```
class DNAseq(Sequence):
    def __init__(self, sequence, Gene_name, Gene_ID, Species_name, Subsp_name):
        super().__init__(sequence, Gene_name, Gene_ID, Species_name, Subsp_name)
        self.AT_content = self.get_ATcontent(sequence)
        self.Transcribed_sequence = self.transcribe_Sequence(sequence)


    def get_ATcontent(self, sequence):
        length = len(sequence)
        A_count = sequence.upper().count('A')
        T_count = sequence.upper().count('T')
        AT_content = (A_count + T_count) / length
        return round(AT_content, 3)
```

```
def transcribe_Sequence(self, sequence):
# transcribe the given DNA sequence into its mRNA sequence and
# store it in the Transcribed sequence instance variable.
    Transcribed_sequence = sequence.replace('T','U')
    return Transcribed_sequence
```

III. Write a subclass of the Sequence class for mRNA sequences named "MRNAseq". It should have the following unique/additional attributes and methods. (15 marks)

Attributes

i. AT_content.
ii. Amino_acid_codons
iii. Translated_sequence

Methods

iv. Constructer method to create DNA sequence objects
v. get_ATcontent(): this should return the AT content of the given sequence and update the AT_content object variable. Because this is for mRNA sequences, rethink the way you should write this method.
vi. upload_Codons(): This class method should create and return a dictionary to store codon-amino acid pairs from a text file
vii. translate_Sequence(): translate a given DNA sequence into its amino acid sequence.

```
class MRNAseq(Sequence):
    Amino_acid_codons = ''

    def __init__(self, sequence, Gene_name, Gene_ID, Species_name, Subsp_name ):
        super().__init__(sequence, Gene_name, Gene_ID, Species_name, Subsp_name)
        self.AT_content = self.get_ATcontent(sequence)
        self.Translated_sequence = self.translate_Sequence(sequence)

    def get_ATcontent(self,sequence):
        length = len(sequence)
        A_count = sequence.upper().count('A')
        T_count = sequence.upper().count('U')
        AT_content = (A_count + T_count) / length
        return round(AT_content, 3)

    @ classmethod
    def upload_Codons(cls, text_file):
        """
        This class method should create and return a dictionary to store codon-amino acid pairs from a
text file
        Input: Argument- text_file
        Output: dictionary which stores codon-amino acid pairs
        """
        cls.Amino_acid_codons = text_file

        #Define an empty dictionary to store codons & amino acid letters
```

```
    Codon_map = {}
    # Open/read codon_table and store it in a variable
    with open(text_file, 'r') as Codon_table:
        for line in Codon_table:
            # Remove header and empty lines in Codon table
            if '#' not in line and line != '\n':
                (codon, amino_acid, Letter, FullName) = line.strip().split('\t')
                # make a dictionary from codons as key and amino acid letters as values
                Codon_map[codon] = Letter
    return (Codon_map)

def translate_Sequence(self, sequence):
    """
    This method is translating a given mRNA sequence into its amino acid sequence.
    (Consider only its first reading frame)
    Input: argument- mRNA sequence
    Output: translated Amino acid sequence
    """
    protein = ''
    #Get codon map from the defined method upload_Codons
    Codon_map = MRNAseq.upload_Codons('codon_table.txt')

    # Define the position of the sequence
    position = 0
    # iterate through the sequence
    while position <= len(sequence)-2:
        # divide the sequence into codons by three bases assuming 1st reading frame
        codon1 = sequence[position:position + 3]

        # go through the dictionary and find relevant amino acid letters
        if codon1 in Codon_map.keys():
            protein += Codon_map[codon1]

        position = position + 3

    # Find the position of aa relevant to start codon and end codon
    AA_No_BeforeMeth = protein.find('M')   #No. of amino acids before Methionine
    AA_No_Before_O = protein.find('O')     #No. of amino acids before the first stop codon

    # Assuming translation is starting with Methionine, get the translated protein seq. and length
    protein1 = protein[AA_No_BeforeMeth:AA_No_Before_O]
    return (protein1)
```

IV. Write a subclass of the Sequence class for protein sequences named “Proteinseq”. It should have the following unique/additional attributes and methods. (10 marks)

Attributes

i. Uniprot_ID
ii. Reviewed_status

iii. Hydrophobicity

Methods

iv. get_Hydrophobicity(): this should return the percentage of the total hydrophobic amino acid residues (A, I, L, M, F, W, Y, V) in the sequence and update the Hydrophobicity object variable.

```
class Proteinseq(Sequence):

    def __init__(self, sequence, Gene_name, Gene_ID, Species_name, Subsp_name, Uniprot_ID,
Reviewed_status):
        super().__init__(sequence, Gene_name, Gene_ID, Species_name, Subsp_name)

        self.Uniprot_ID = Uniprot_ID
        self.Reviewed_status = Reviewed_status
        self.Hydrophobicity = self.get_Hydrophobicity(sequence)

    def get_Hydrophobicity(self, sequence):
        """
        this method should return the percentage of the total hydrophobic amino acid residues
        (A, I, L, M, F, W, Y, V) in the sequence & update the Hydrophobicity object variable

        Input: argument- protein sequence
        Output: percentage of the total hydrophobic AA residues
        """
        count = 0
        hydrophobic_AA = ['A','I','L','M','F','W','Y','V']

        for letter in sequence:
            if letter in hydrophobic_AA:
                count += 1
                hydrophobicity = (count / len(sequence))*100
            else:
                hydrophobicity = 0

        return round(hydrophobicity,2)
```

V. Write a Python program to read the sequences in OSDREB_sequences.FASTA file and create objects for each FASTA record. Moreover, perform the following tasks using the Sequence class. You can write this script in a separate file and import the Sequence class to perform the tasks. When creating objects, you can manually type each parameter for the object necessary for running the following commands or you can pass a list of elements as parameters using the following command. Use sequence name as the object name.
Object_name = Class_name(*[a list of parameters to be passed in the correct order])
(25 marks)

i. Print the following details for the OSDREB1A DNA sequence: Gene ID, sequence length, sequence type and AT content.

ii. Transcribe the OSDREB2B coding sequence and create a new object for the resulting mRNA sequence. Print the length and sequence type, AT content, and the sequence of the resulting mRNA sequence

iii. Translate the OSDREB2B mRNA sequence created above into its amino acid sequence and print the result and also print its length.

iv. Print the Uniprot ID, reviewed status, type, amino acid composition and the Hydrophobicity of DREB2A protein.

v. Output the number of sequences created using the Sequence class variable.

```
from lab3_s13722_Q2 import *

seq_set = Sequence.fasta_Split("OSDREB_sequences.fasta")
print(seq_set)

DREB1A_P = seq_set['DREB1A P']
DREB1B_P = seq_set['DREB1B P']
DREB2A_P = seq_set['DREB2A P']
DREB2B_P = seq_set['DREB2B P']
DREB1A_CDS = seq_set['DREB1A CDS']
DREB1B_CDS = seq_set['DREB1B CDS']
DREB2A_CDS = seq_set['DREB2A CDS']
DREB2B_CDS = seq_set['DREB2B CDS']

#Creating objects manually from the seq_set dictionary
obj_DREB1A_P =Proteinseq(
DREB1A_P[0],DREB1A_P[1],DREB1A_P[2],DREB1A_P[3],DREB1A_P[4],DREB1A_P[5],
DREB1A_P[6])
obj_DREB1B_P =Proteinseq(
DREB1B_P[0],DREB1B_P[1],DREB1B_P[2],DREB1B_P[3],DREB1B_P[4],DREB1B_P[5],
DREB1B_P[6])
obj_DREB2A_P =Proteinseq(
DREB2A_P[0],DREB2A_P[1],DREB2A_P[2],DREB2A_P[3],DREB2A_P[4],DREB2A_P[5],
DREB2A_P[6])
obj_DREB2B_P =Proteinseq(
DREB2B_P[0],DREB2B_P[1],DREB2B_P[2],DREB2B_P[3],DREB2B_P[4],DREB2B_P[5],
DREB2B_P[6])

obj_DREB1A_CDS = DNAseq
(DREB1A_CDS[0],DREB1A_CDS[1],DREB1A_CDS[2],DREB1A_CDS[3],DREB1A_CDS[
4])
obj_DREB2B_CDS = DNAseq
(DREB2B_CDS[0],DREB2B_CDS[1],DREB2B_CDS[2],DREB2B_CDS[3],DREB2B_CDS[4
])
```

```
print('i.')
print('Gene ID :', obj_DREB1A_CDS.Gene_ID)
print('sequence length :', obj_DREB1A_CDS.Seq_length)
print('sequence type :', obj_DREB1A_CDS.Seq_Type)
print('AT content :',obj_DREB1A_CDS.AT_content ,'\n' )

print('ii.')
mRNA_sequence = obj_DREB2B_CDS.Transcribed_sequence
obj_DREB2B_mRNA = MRNAseq( mRNA_sequence,
DREB2B_CDS[1],DREB2B_CDS[2],DREB2B_CDS[3],DREB2B_CDS[4])

print('sequence length :', obj_DREB2B_mRNA.Seq_length)
print('sequence type :', obj_DREB2B_mRNA.Seq_Type)
print('AT content :',obj_DREB2B_mRNA.AT_content)
print('Transcribed_mRNA sequence of OSDREB2B:',
obj_DREB2B_CDS.Transcribed_sequence,  '\n')

print('iii.')
Protein = obj_DREB2B_mRNA.Translated_sequence
print('Translated_sequence of OSDREB2B mRNA :',Protein)
print('length :', len(Protein), 'aa', '\n')

print('iv.')
print('Uniprot_ID :', obj_DREB2A_P.Uniprot_ID)
print('Reviewed status :', obj_DREB2A_P.Reviewed_status)
print('Type :',obj_DREB2A_P.Seq_Type)
print('amino acid composition :', Sequence.get_Character_Count(DREB2A_P[0]))
print('Hydrophobicity ;', obj_DREB2A_P.Hydrophobicity , '\n')

print('v.')
print('number of sequences created :', Sequence.sequence_count)
```