S13674
2017s16415
D.P.B.Hendalage

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

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).*

   DREB1A→ Q64MA1→ XP_015610912→reviewed
   DREB1B→ Q3T5N4→ XP_015610928→reviewed
   DREB2A→ Q0JQF7→ XP_015621339→reviewed
   DREB2B→ Q5W6R4→XP_015639655→reviewed

   II. Write the equation and an algorithm to calculate the AT content of a given DNA or mRNA sequence. *(4 marks)*
   Find mRNA from amino acid sequences and mRNA sequence
   Define a base counter
   Count AT bases
   Give output

```
Create a method for count AT bases
```

```
            check and select DNA
            count the AT bases
            give the output
    method for split fasta file
            define an empty dictionary
            read the file
            iterate through each line
            check the new line character
               strip the other lines
            check for the header lines
            assign key as the dictionary
            then other striped lines as assign as values
            return the output
    method for find type of the molecule
           check for "M" in sequence for aa
           check for "U" in sequence and "M" not in sequence for mRNA
           then others take as DNA
```

```
'''
Author : Bhagya Hendalage
Date : 27/12/2020
input = sequence
output = AT count
'''
def AT_content(sequence):
    #define counters for 'A' and 'T'
    A=0
    T=0
    for base in sequence:
        if base=='A':
            A+=1
        elif base=='T':
            T+=1
    print("A base count is:",A)
    print("T base count is:",T)
#using custom sequence
AT_content('ATATATACGCGCG')
```

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)*

```python
#method for count AT bases
def AT_counts(sequence):
    AT_count=0
    #check and select DNA
    if 'M' not in sequence:
     for i in sequence:
         if i=='A' or i=='T':
             AT_count +=1
    #if sequence is an amino acid
    else:
        AT_count = "Not a DNA sequence"
    return  AT_count
```

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)*

```python
#method for split fasta file
def split_fasta_file(filename):
    #define an empty dictionary
    dict = {}
    with open(filename, 'r') as file:
        header = ''
        for line in file:
            if line != '\n':
                line = line.strip()
                # identify the headers
                if line.startswith('>'):
                    header = line
                    dict[header] = ''
                # if it is not a header, it defines as sequence
                else:
                    dict[header] += line.strip()
    print(dict)
    return (dict)
```

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

```
#method for find type of the molecule
def type_of_molecule(sequence):
        if 'M' in sequence:
            return "Amino acid"
        elif 'U' in sequence and 'M' not in sequence:
            return "mRNA"
        else:
            return "DNA"
```

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

```
#give an input to a method and output save as a variable
dic = split_fasta_file("OSDREB_sequences.fasta")
for key,value in dic.items():
    print('header : ',key)
    print('sequence : ',value)
    print('Type Of the Molecule : ',type_of_molecule(value))
    print('AT count in the sequence : ',AT_counts(value), '\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.

```
class Sequence:
    gene_ID=''
    gene_name=''
    seq_type=''
    seq_length=0
    seq_count=0
    sp_name=''
    subsp_name=''
    sequence=''
#define attributes and methods
    def __init__(self,gene_name,gene_ID,sp_name,subsp_name,sequence):
        self.gene_ID=gene_ID
        self.gene_name=gene_name
        self.sp_name=sp_name
        self.subsp_name=subsp_name
        self.sequence=sequence
        self.seq_type=Sequence.get_Seq_Type(sequence)
        self.seq_length=len(sequence)
        Sequence.seq_count += 1

    @staticmethod
    def fasta_Split(file_name):
        """
        method to split fasta file to dectionary
        For proteins:{'Gene_name':(['>Gene_name_P' ,'RefSeq_gene_ID','species','subspecies','Uniprot_ID','reviewed_status'],'sequence')}
        For coding sequences:{'Gene_name:(['>Gene_name','CDS-RefSeq_gene_ID','species','subspecies'],'sequence')}
```

```
        method to split fasta file to dectionary
        For proteins:{'Gene_name':(['>Gene_name_P' ,'RefSeq_gene_ID','species','subspecies','Uniprot_ID','reviewed_status'],'sequence')}
        For coding sequences:{'Gene_name:(['>Gene_name','CDS-RefSeq_gene_ID','species','subspecies'],'sequence')}

        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 '-'
                        Gene_name = line.split('-')
                        #to take the first index value as gene_name
                        Gene_name = Gene_name[0]
                        header_parts = line.split('-')
                        #to take all indeces to value
                        header_parts = header_parts[0:]
                        #without seq='', all keys contain first iterated sequence
                        seq=''
                    if '>' not in line:
                        seq = seq + line
```

```
                    seq = seq + line
                    dict[Gene_name] = header_parts, seq
    return(dict)

@staticmethod
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"

@staticmethod
def get_Character_Count(sequence):
    """
    method for return all characters in sequence and its count as a dictionary
    Input: sequence
    Output: dictionary of character counts
    """
    character_count = {}
```

```
    character_count = {}

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

    # printing result
    return(character_count)
```

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.

```
#subclass for DNA
class DNAseq(Sequence):
    AT_content=0
    Transcribed_sequence=''


    def __init__(self, gene_name, gene_ID,sp_name, subsp_name,sequence):
        super().__init__(gene_name, gene_ID, sp_name, subsp_name, sequence)

        self.AT_content = self.get_AT_Contents(sequence)
        self.Transcribed_sequence=self.transcribe_Sequence(sequence)



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


    def get_AT_Contents(self,sequence):
        '''
        method for get A,T bases content in sequence as a percentage
```

```
    def get_AT_Contents(self,sequence):
        '''
        method for get A,T bases content in sequence as a percentage
        Input : sequence
        Output : AT content as percentage
        '''
        #store the output as instance variable
        AT_content = ((sequence.count("A") + sequence.count("T")) / len(sequence)) * 100
        return AT_content
```

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.

```python
#subclass for mRNA
class mRNAseq(Sequence):
    AT_content=0
    Amino_acid_codons=''
    Translated_sequence=''


    def __init__(self, gene_name, gene_ID,sp_name, subsp_name,sequence):
        super().__init__(gene_name, gene_ID, sp_name, subsp_name, sequence)

        self.mRNA_id = gene_ID
        self.mRNA_name = gene_name
        self.seq_type = "mRNA"
        self.seq_len = len(sequence)
        self.AT_content = self.get_ATcontent(sequence)
        self.Translated_sequence = self.translate_Sequence(sequence)


    def get_ATcontent(self,sequence):
        '''
         methode for get AT content in mRNA sequnce
         input : sequence
         output : AU content as a percentage
        '''
        AT_content = ((sequence.count("A") + sequence.count("U")) / len(sequence)) * 100
        return AT_content
```

```
def upload_Codons(mRNAseq,CodonfileName):
    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, sequence, CodonfileName="OSDREB_sequences.fasta"):
    protein = ''
    Codon_map = self.upload_Codons(CodonfileName)
    #start when meet 'AUG'
    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)
```

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.

```python
#subclass for proteins
class Proteinseq(Sequence):
    Uniprot_ID=''
    Reviewed_status=''
    Hydrophobicity=0
    def __init__(self, gene_name, gene_ID, sp_name, subsp_name, sequence,uniprot_ID,reviewed):
        super().__init__(gene_name, gene_ID, sp_name, subsp_name, sequence)
        self.prot_name=gene_name
        self.prot_ID=gene_ID
        self.uniprot_ID=uniprot_ID
        self.reveiwed=reviewed
        self.Hydrophobicity=self.get_Hydrophobicity(sequence)

    def get_Hydrophobicity(self,sequence):
        '''
        method for find hydrophobicity using content of hydrophobic amino acids
        input : sequence
        output : percentage of hydrophobicity
        '''
        Hydrophobicity = ((sequence.count("A") + sequence.count("I") + sequence.count("L") + sequence.count(
            "M") + sequence.count("F") + sequence.count("W") + sequence.count("Y") + sequence.count("V")) / len(
            sequence)) * 100
        return(Hydrophobicity)
```

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])
(25marks)

   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 s13674_PR3_Q2 import *
sequence=Sequence.fasta_Split("OSDREB_sequences.FASTA")
seq_objects = []

for key,value in sequence.items():
    if Sequence.get_Seq_Type(value[1])=="DNA":
        seq_objects.append(DNAseq(key,value[0][1],value[0][2],value[0][3],value[1]))
    elif Sequence.get_Seq_Type(value[1]) == "mRNA":
        seq_objects.append(mRNAseq(key,value[0][1], value[0][2], value[0][3],value[1]))
    else:
        seq_objects.append(Proteinseq(key,value[0][1],value[0][2],value[0][3],value[1],value[0][4],True))
for objects in seq_objects:

    if objects.seq_type == "DNA" and "DREB1A" in objects.gene_name:
        print('i')
        print("Gene ID: ",objects.gene_ID)
        print("Sequence length: ", objects.seq_length)
        print("Sequence type: ", objects.seq_type)
        print("AT content: ", objects.AT_content,'%')
        print()

    if objects.seq_type == "DNA" and "DREB1B" in objects.gene_name:
            print('ii')
            new_seq = DNAseq.transcribe_Sequence(DNAseq, objects.sequence)
            mRNA = mRNAseq(objects.gene_name, objects.gene_ID, objects.sp_name, objects.subsp_name, new_seq)
            print("Sequence length: ", mRNA.seq_len)
            print("Sequence type: ", mRNA.seq_type)
```

```
    if objects.seq_type == "DNA" and "DREB1B" in objects.gene_name:
            print('ii')
            new_seq = DNAseq.transcribe_Sequence(DNAseq, objects.sequence)
            mRNA = mRNAseq(objects.gene_name, objects.gene_ID, objects.sp_name, objects.subsp_name, new_seq)
            print("Sequence length: ", mRNA.seq_len)
            print("Sequence type: ", mRNA.seq_type)
            print("AT content: ", mRNA.AT_content,'%')
            print("Sequence: ", mRNA.sequence)
            print()

            print('iii')
            prot = mRNA.Translated_sequence
            print("protein: ", prot)
            print("length: ", len(prot))
            print()

    if objects.seq_type == "Amino acid" and "DREB2A_P" in objects.gene_name:
            print('iv')
            print("Uniprot ID: ", objects.uniprot_ID)
            print("reviewed: ", "yes" if objects.reveiwed else "no")
            print("Composition: ", objects.get_Character_Count(objects.sequence))
            print("hydrophobicity: ", objects.Hydrophobicity,'%')
            print()
print('v')
print(Sequence.seq_count)
```