Project Documentation

(Edoardo Trastu, Luis Alberto Espinoza Garcia, Matilde Menini)

**Introduction to main goal**

Our DNA Sequence Analysis Program is a simple yet powerful, tool which is designed to provide a comprehensive analysis of genetic information including sequence manipulation. This program is created to meet the needs of bioinformatics, researchers, geneticists and anyone interested in DNA sequencing.

**Classes:**   
Constructing clear-structured classes and the program web-visualization is our focus to achieve a well-structured program with an accessible code. From the Fasta file, the program constructs a DNA object or a AAchain object, encapsulating the essential attributes and behaviours associated with sequences. The program uses array programming to speed the operations. Here’s a quick review of the structural and functionality choices applied to the classes:

The file “read\_fasta.py” is used to read the Fasta files and it contains the Class Dataset.

**Class Dataset:**

The class Dataset is found in the file read\_fasta.py, it is used to instantiate objects that contain all the information needed to instantiate Sequences objects.

-The attribute “self.data” is a pd.Series containing as elements a series of nucleotides or amino acids represented as single characters strings

-The attribute “self.DNA\_RNA” is a Boolean that if positive it means that nucleotides are present in the series, if False are present amino acids, this is needed because nucleotides and amino acids share elements in series, like ‘A’ in nucleotides is adenine, but in amino acids represent alanine

+The method “readfasta\_seq(fasta:dir,DNA\_RNA:bool)” is a method that resets both the previously explained attributes, by reading a Fasta file (.fa , .fasta , .fna), as argument has a directory to the Fasta file and the new Boolean to put in the attribute “Self.DNA\_RNA”. The program opens the Fasta file and saves and skips the first row, because it contains the name of the file, not the sequence. Then saves the whole text as string and eliminates the \n and if the Fasta file is composed by many chromosomes or accessory genomes, it will consider just the main strand by slicing it until the first ‘>’ present, excluding the first row, everything that follows the ‘>’ is the accessory genome’s name and accessory genome itself. Some Fasta files have a mix of lowercase and uppercase characters, so it’s fundamental to turn everything uppercase. Now the obtained string has the correct sequence, list(string) is used to separate each character, then turn it into a pd.Series and save the name of the file in the pd.Series. Now the value of “self.data” is re-set as the pd.Series created, also the value of “self.DNA\_RNA” is re-set.

+The method “get\_data” returns the attribute “self.data”.

+The method “get\_DNA\_RNA” returns the attribute “self.DNA\_RNA”.

The file “main” contains the classes of the “Genetic Entities” and its subclasses, the many classes are instantiated with Dataset class in the “read\_fasta.py” file.

**Class Genetic Entities:**  
The class “Genetic Entities” is the superclass of all the following classes, it’s an empty class.

**Class OrganicElements:**

The class “OrganicElements” is a subclass of “Genetic Entities”.

-The attribute “self.letter” is a string that represent the symbol of a nucleotide or amino acid.

+The method “get\_letter()” returns “self.letter”, a string.

+The method “set\_base()” modifies the attribute “self.letter”.

**Class Nucleotide:**  
It’s a subclass of “Organic Elements”, these are the units used to build the DNA and mRNA strand.

**Class AA:**  
It’s a subclass of “Organic Elements”, it is used this this class to build the amino acid chain.

**Class Sequences:**  
The class “Sequences” is a subclass of “Genetic Entities”. The class constructor is fed as arguments the initial Dataset, which is a pandas Series, containing as elements character strings representing or nucleotides or amino acids, and a Boolean value DNA\_RNA which allows to distinguish the different sequence input types (DNA sequence or Amino Acid sequence). The constructor initializes the following objects:

-The attribute “self.name” is the name of the series.

-The attribute “self.strand”, given the series, it turns the series elements into Organic Elements.

-The attribute “self.DNA\_RNA” is a Boolean value.

-The attribute “self.blocks\_letters” is a list containing the unique characters of the Series.

-The attribute “self.building\_blocks” is a list created from “self.blocks\_letters” and “self.DNA\_RNA”. It uses the first attribute for the position and for the content of the organic element and the second attribute to select which type of organic element to instantiate. Finally, it’s going to be obtained a list which contains the correspondent organic elements of “self.block\_letters” ,egs. DNA\_RNA= True, [‘A’,’T’,’G’,’C’] --> [nucleotide(‘A’), nucleotide(‘T’), nucleotide(‘G’), nucleotide(‘C’)], if DNA\_RNA= False would use AA() instead of nucleotide().

-The attribute “self.blocks” is a dictionary that associates the elements of“self.blocks\_letters” together with “self.building\_blocks”, then uses this to replace the strings of “self.strand” into Organic Elements objects. In this way, for a genome of 1 million of bases, only the 4 organic elements contained in “self.building\_blocks” are needed.

+The method “get\_name(self)” returns “self.name”.

+The method “get\_strand(self)” returns “self.strand”.

+The method “mantein\_organic\_elemenst(self)” uses the attribute “self.blocks” to access the Organic elements and reset them back to their original value if they were modified.

+With the method “transcription(self)”, if the object that uses this method is an instance of DNA, self.strand is considered as the strand containing the genetic information, so the complementarity strand must be biologically transcribed, this is another complementarity and generates an RNA strand, so a “U” nucleotide is obtained instead of a “T”. Because there is a double complementarity, there is the need to just modify the nucleotide “T”, using the “self.blocks” dictionary, where the instance of nucleotide is accessed and associated to ‘T’ and, by applying the function “.set\_base(‘U’)” (organic element method), the new strand for the mRNA is rapidly obtained, but “self.strand” can’t be used to instantiate it, so its elements are turned into strings and then instantiate mRNA using that Series and it will create its own Organic elements. At the end “mantein\_organic\_elements()” is used and then return the mRNA. If the object is a mRNA or AAchain type it will return itself, not modifying anything.

+The class method “gen\_data(cls, ser)” accepts any Series uses pandas functions to get the data of the Series(counts, frequencies, min, max) into a list of strings, a matrix containing counts and frequencies is obtained.

+The method “produce\_graph” uses a Series which is turned into a smaller Series using .value\_counts() and creates a bar plot with “plot(kind = “bar”)” displaying the Oragnic elements of the x axis with “plt.xlabel” ("organic elements") and their respective frequency with “plt.ylabel” (“frequency”).

+The class method ”turn\_in\_str(cls,ser)” allows to turn Series, that have as elements strings or objects which can be turned into strings, to generate a single string containing the sum of all the elements.

**Class DNA(Sequences):**

DNA is a subclass of “Sequences”, it has the same attributes of its parent class.

+The method “produce\_negative\_strand()” uses “self.blocks” to set the complementarity bases of “self.strand”. It produces another Series that has its elements as strings (astype(str)) and reverses it using “.iloc[: :-1]”. After, the original organic elements are setted back with the method “.mantein\_organics\_elements()” and the previous Series is returned.

**Class mRNA (Sequences):**

The class mRNA is a subclass of Sequences, it has the same attributes of its parent class.

+The method “produce\_negative\_strandRNA()” is the same function appearing in the DNA class, but instead of having ‘T’ has ‘U’.

+The method “translation()” serves to identify all amino acids from our strand and return all possible amino acid chains. It is obtained a copy of “self.strand” that has as elements strings, other reading frames are created with “.iloc[1::]” and “.iloc[2::]”. In both Series the index is resetted, now these 3 Series are summed to obtain all the codons of all the reading frames. Some NaN are obtained and are dropped with “.dropna()”. Now it’s obtained the same Series, but instead of using “self.strand”, it is used the series that is obtained from “.produce\_negative\_strandRNA()”. Now in the 2 Series containing the positive and negative RFs are replaced the codons with amino acids by using “.map(dict)” that has as dict a dictionary of the genetic code. A map is preferred compared to “.replace” because if it doesn’t find an association it will insert a “NaN” value. The non codified codons are due to the presence of uncertain nucleotides, like “N” that means that any nucleotide can be found in that position. The uncertain codons are replaced with a ‘?’ with the method fillna(‘?’) which easily recognise NaN values. The 2 series then are sliced to produce the Series to instantiate all the AAchain objects. The method returns them into values of a dictionary and the keys are their respective reading frame (a number).

Initially in “codons” we obtain all the possible codons in all reading frames in the positive strand in a list which will be dividing by splicing later to differentiate all RF. In “negstrand”, the negative DNA strand is taken from the and transcribed into mRNA and by using “self.blocks[“U”].set\_base(“A”)”, which once again allows to easily modify a base. Two series arrays containing respectively the positive strand (all RF) codons and the negative strand (all RF) codons are initialized in “codons” and “codonsneg”. “GeneticCode” contains a dictionary with all codons with associated amino acids. In “codons” and “codonsneg”, which are lists containing codons as elements, each element codon inside is replaced with its amino acid by using “codons.replace(GeneticCode)” and “codonsneg.replace(GeneticCode)”. Each different RF is returned (by splicing) under the form of a series which contains an amino acid each row. A dictionary with each RF number (negative and positive strand) is associated to its series. Six series (three from positive and three from negative) are returned.

**Class AAChain:**  
In the class “AAChains”, which is subclass of the parent class “Sequences”, all the possible amino acid chains (proteins or oligopeptides) are identified. Once again all “Sequence” class methods are overridden. In the Numpy arrays “self.starts” and “self.stops” all indexes of respectively start and stop codons are found with the function “np.where()” and stored. The following “if” conditional branching allows the pairing up of each start codon with the next (closer) stop. Branching with “while” and “if” conditions is key to allow the finding of all types of protein-coding sequences, meaning that there can be multiple starts for a single stop and also these shorter peptide chains should be considered. In “self.aachains” attribute, a pandas data frame with two columns: the first displaying all amino acid chains in the form of lists and in the second column the length of each codon list. These rows are sorted in ascending length order.

In the method “get\_oligos()”, the value “ind” is used to find the first element in amino acid chains that has 20 amino acids and store its index. All indexes from 0 to “ind” of “self.aachains” data frame are returned; hence all the oligopeptides are returned.

In the method “get\_proteins”, the value “ind” is used to find the first element in amino acid chains that has 20 amino acids and store its index. All indexes from “ind” to the end of “self.aachains” data frame are returned; hence all the proteins are returned.

In the method “get\_single\_aachain(ind)”, “ind” is used to find and return the specific Series of the protein or oligo present in that position in self.aachains.

In the method “get\_single\_prot\_len()” we use “ind” to find and return the length of the protein or oligo present in that position in “self.aachains”.

**Web visualization:**

This part of the program is a web application built using the Flask framework. This program provides functionalities related to DNA sequencing and analysis. Users can view the generated DNA strand, its negative strand, RNA sequences, translated amino acid chains, and derived oligopeptides and proteins. There is also functionality to visualize the DNA sequence and individual proteins. Overall, this program provides a comprehensive platform for analysing DNA sequences and their resulting biological products, such as RNA and proteins, through a user-friendly web interface.  
The Flask application is initialized, and an instance of Flask is created and the attribute “myDNAClass”, “app.myRNAClass” and “app.myAAChainDict” are defined and set to None.

The function “home” renders the home page template “home.html”. This is the landing page of the application where users can upload a DNA sequence fasta file or an Amino Acid sequence fasta file.

The function “setfile” handles the upload functionality of DNA sequences, in fact, it checks if a file is uploaded and if is correct meaning it is a .fasta or a .fa or a .fna file (if not it returns “home.html”), creates two accessible directories: “fastas” and “static”, in the directory “fastas” a copy of the Fasta file is saved, while the other remains empty. The function sets “myDNAClass” as attribute to the DNA sequence read from the uploaded file. Every page except for “home”, “singleprotein” and “singleoligopeptide” display the DNA name as the title.

The function “setfile1” handles the upload functionalities of amino acid sequences. It works just like the function “setfile”, however it redirects directly to the page “translate\_single” since transcription and translation aren’t required.

Since all fasta files that contain a protein miss the stop codon, the function adds one.

In normal circumstances an amino acid sequence represents just one protein, hence the best way to visualize it is to represent just one page which contains the whole data.

The function “choice” allows to obtain two links which allows the user to open the positive or the negative strand which will respectively bring to the web page “generate.html” or the web page “generate-.html”.

The function “generate” renders a template to display the generated positive DNA strand from the uploaded file. If the accessibility to the file “app.myDNAClass” is lost, the function returns the “home.html” page otherwise allows the generation of the positive DNA strand in the form of string. This branching condition will be repeated in each function of the code as a precaution to prevent file malfunctioning.

The “generate-” function works the same as generate function, however it renders a template to display the negative strand of the DNA sequence. The attribute “dna” contains the DNA negative strand and “dna\_str” contains the DNA negative sequence in str form.

The “visualize” function renders a template to display the visualization of the nucleotide frequency plot of the chosen sequence. The plot is saved in the “static” directory. The attribute “data” contains a list with all the statistics about the sequence. The attribute “app.myRNAClass” is set with the DNA transcription.

The many “transcribe” functions render a template to display the transcribed RNA sequences from the DNA sequence. The attributes RNA\_pos and RNA\_neg contain respectively the RNA positive strand and RNA negative strand, while RNA\_pos\_str and RNA\_neg\_str contain RNA\_pos and RNA\_neg turned into strings.

The function “translate” render the template to display all the six reading frames and sets the attribute “app.myAAChainDict” to contain the RNA translated chain.

The six successive “translate\_n” functions render the respective templates to display each reading frame. The attribute “AAChainstrand” gets each amino acid chain and turns it into string.

The function “translate\_single” renders a template to display the visualization of the amino acid frequency plot of the respective reading frame. The plot is saved in the “static” directory. The attribute “data” contains a list with all the statistics about the amino acid sequence.

The function “oligopeptides” renders a template to display information about proteins derived from the translated RNA sequences. A for loop is initialized to create another data frame where all reading frames are stored. This data frame contains another column with associated reading frame number.

The function “proteins” renders a template to display information about proteins derived from the translated RNA sequences. A for loop is initialized to create another data frame where all reading frames are stored. This data frame contains another column with associated reading frame number.

The function “singleoligopeptide” renders a template to display information about a single oligopeptide. Reading frame and index request respectively the orf key and the index number that the user select in the oligo.html page. The orf is used as a key for dictionary that contain the AAchain. The index is used by ‘get\_single\_prot\_len' and ‘get\_single\_aachain’ to obtain the specific length and chain of a precise AAchain. “plt1” produces a graph using chain as series the graph is saved as png in the folder “static”.

The function “singleprotein” renders a template to display information about a single protein. Reading frame and index request respectively the orf key and the index number that the user select in the “proteins.html” page. The orf is used as a key for dictionary that contain the AAchain.The index is used by ‘get\_single\_prot\_len' and ‘get\_single\_aachain’ to obtain the specific length and chain of a precise AAchain. “plt1” produces a graph using chain as series the graph is saved as png in the folder “static”.

**CRC cards:**

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **SUPERCLASS:** | | **SUBCLASSES:** |
| Dataset | Object | | \ |
| **Responsabilities:** | | **Collaborations:** | |
| Read the fasta file | | \ | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **SUPERCLASS:** | | **SUBCLASSES:** |
| Genetic entities | Object | | Sequences, Organic Elements |
| **Responsabilities:** | | **Collaborations:** | |
| \ | | \ | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| Organic elements | Genetic entities | | Nucleotides, AA |
| **Responsabilities:** | | **Collaborations:** | |
| 1. -return and set the base/aa type | | 1. \ | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| Nucleotides | Organic Elements | | \ |
| **Responsabilities:** | | **Collaborations:** | |
| - Distinguish nucleotide objects from amino | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| AA | Organic elements | | Proteins, Oligopeptides |
| **Responsabilities:** | | **Collaborations:** | |
| - Defines and separates amino acid objects from nucleotide objects | |  | |

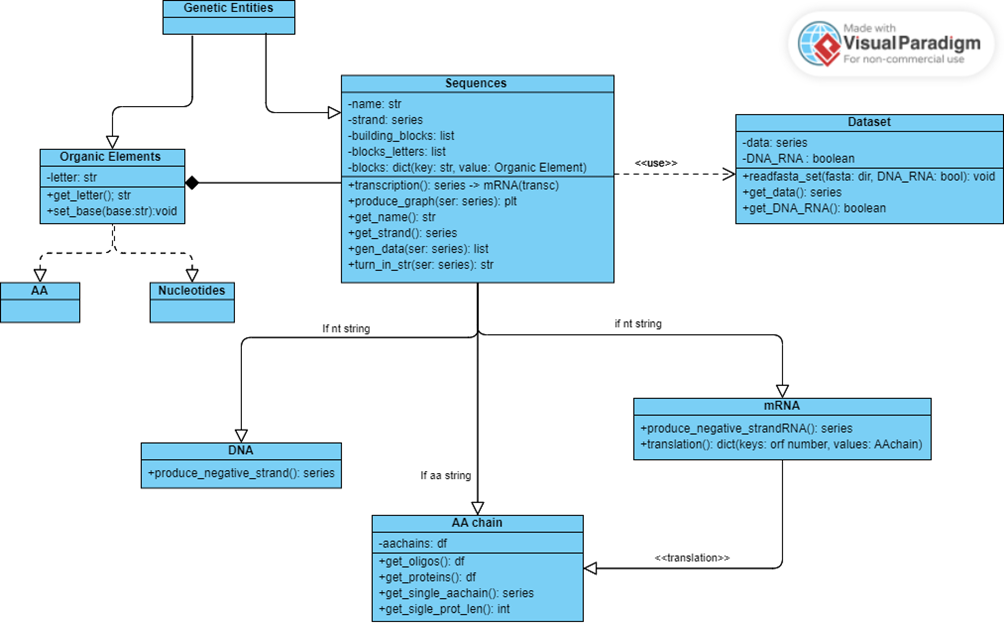
|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| Sequences | Genetic entities | | DNA, RNA, AAchain |
| **Responsabilities:** | | **Collaborations:** | |
| 1. Associates Datasets elements with elements organic\_elements 2. Able to mantein the values of organic elements 3. Able to transcribe DNA instances and intanciate mRNA 4. Produce graph of the elements of a series panda 5. Produce data, min , max, frequencies of series 6. Able to turn the series into a string | | 1. DNA, AAchain, AA, Nucleotide 2. AA, Nucleotide 3. DNA, mRNA, Nucleotide 4. \ 5. \ 6. \ | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| **DNA** | Sequences | | \ |
| **Responsabilities:** | | **Collaborations:** | |
| 1. Perform transcription of the vector containing nucleotides  2. Obtain nucleotide frequency  3.produce negative strand | | 1. Sequences, nucleotide, mRNA 2. Sequence 3. Sequence, Nucleotide | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| mRNA | Sequences | | \ |
| **Responsabilities:** | | **Collaborations:** | |
| 1. Produce negative strand   2. Perform translation, using all the reading frames, each reading frame produces a AAchain | | 1. Sequence, nucleotide, mRNA 2. Nucleotide, Sequences, AA, AAchain | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS NAME:** | **Superclass:** | | **Subclasses:** |
| AA chain | Sequences | | Proteins, oligopeptides |
| **Responsabilities:** | | **Collaborations:** | |
| 1.Individuate and sort all possible proteins and oligopeptides  2.Allows to select a single one | | 1.\  2.\ | |

**UML diagram:**



**Description of UML diagram:**

The class Genetic Entities is an abstract empty superclass.   
Sequences use the Dataset class to read the Fasta file and obtain a sequence and a Boolean value. The Boolean value allows the identification of a nucleotide sequence (True) or of an amino acid chain (False).   
The class Sequence strongly depends upon the Organic Elements class as each nucleotide or amino acid is instantiated with a specific Organic Element object which belongs either to the class Nucleotides or the class AA. In the case of obtaining an amino acid chain, it is going to be inherited and handled in the class AAChain. However, if the sequence is a nucleotide DNA array, the DNA negative strand is going to be produced in the class DNA, it is also going to be transcribed to obtain an RNA strand in the Sequence class and it will be further inherited and handled by the class mRNA. In the mRNA class, the RNA strand is going to be translated into amino acid chains and each chain will be inherited and handled by the class AAChain. In AAChian these amino acid chains are going to be divided between oligonucleotides and proteins.