Computational Bioinformatics

Computational Bioinformatic Assignment

Author: Zirou Wei [ziw11] Student

Number:249051751

Module:CS31420

12/12/2025

1.Introduction

1.1 Project Overview and objective

As identified in the preliminary analysis, the current bases found on Mar differ significantly from the standard terrestrial genetic code (DNA bases A, T, G, C and RNA bases A, U, G, C). Thus, the existing terrestrial genetics no longer apply to Martian organisms, and all corresponding mechanisms need to be re-decoded. And it is worth noting that the final goal is to decipher Gene b's mRNA sequence and its protein sequence. The Gene a sequence, along with its mRNA sequence and protein sequence serves as a reference for decoding to decipher Martian genetic language. To achieve this, I will:

1.Identify the basic bases of Martian DNA and mRNA

2.Determine the base-pairing relationship between Martian DNA and mRNA

3.Calculate the codon length

4.Construct a codon-to-amino acid lookup table

These objectives align with Task 1-4. Task 5 will then involve invoking functions from other modules to get the Martian genetic rules, which will be used to decode the mRNA and protein sequences of Gene b.

The biological principle inside it is as follows: we first use the detected Martian DNA bases to transcribe mRNA(following the complementary base-pairing rule). To determine the codon length, we apply the formula:

$$\text{Codon Length} = \frac{\text{Total Length of mRNA}}{\text{Total Number of Amino Acids}}$$

Once the codon length is confirmed, we can segment the mRNA sequence into individual codons, then deduce the codon-to-amino acid correspondence. Finally, this correspondence will be used to predict the protein sequence of Gene b.

2.Implementation, Challenges, and Results of 5 Coding Tasks

2.1Task1—— Quantification and Visualization of Base Data

The purpose of task1 is to identify the symbols on the Mars to its DNA and RNA sequence. After that, visualize and quantify them.

2.1.1 Problem Analysis and Logic

The Challenge:

The primary challenge in Task 1 is that the genetic symbols on Mars are unknown. Unlike standard terrestrial genetics where we clearly know the base types（DNA: A, T, C, G; RNA: A, U, C, G）, Martian biology is completely different. Therefore, the analysis logic cannot be based on simple sequence counting with the pre-known base pairs

The Approach:

To solve this, I put each newly appearing base in the sequence into a list of discovered bases. The subsequent analysis involves two key steps:

1.Quantification: Counting each new base and dividing it by the total sequence length to calculate the proportion.

2.Visualization: Presenting these appearing base frequencies via a bar chart.

Technical Barrier:

A specific difficulty I encountered was my lack of prior experience with Python visualization libraries. To overcome this, I conducted self-study on the matplotlib library, concerning canvas setup, axis data preparation, and adding descriptive labels to ensure the analysis results could be exported or saved effectively.

2.1.2 Function modules and Implementation

The structure of the code has one calling function and three core processing functions.

Step 1: Data Cleaning(read_file_sequence)

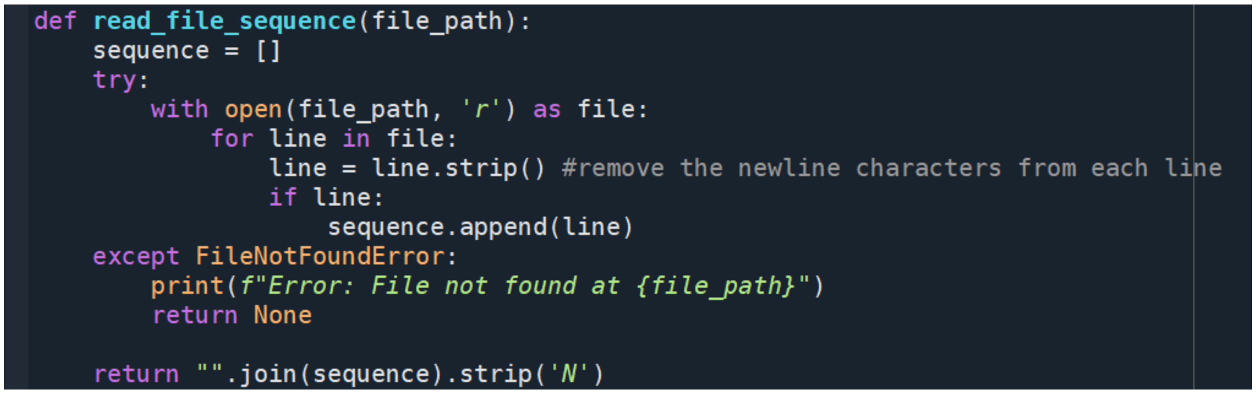
1.Problem: Raw file data contains newline characters and placeholder ‘N’ symbols in the beginning and ending of sequence.

2.Cleaning process:

First, I iterate through each line and use strip() to remove hidden newline characters, then append() lines to a list.

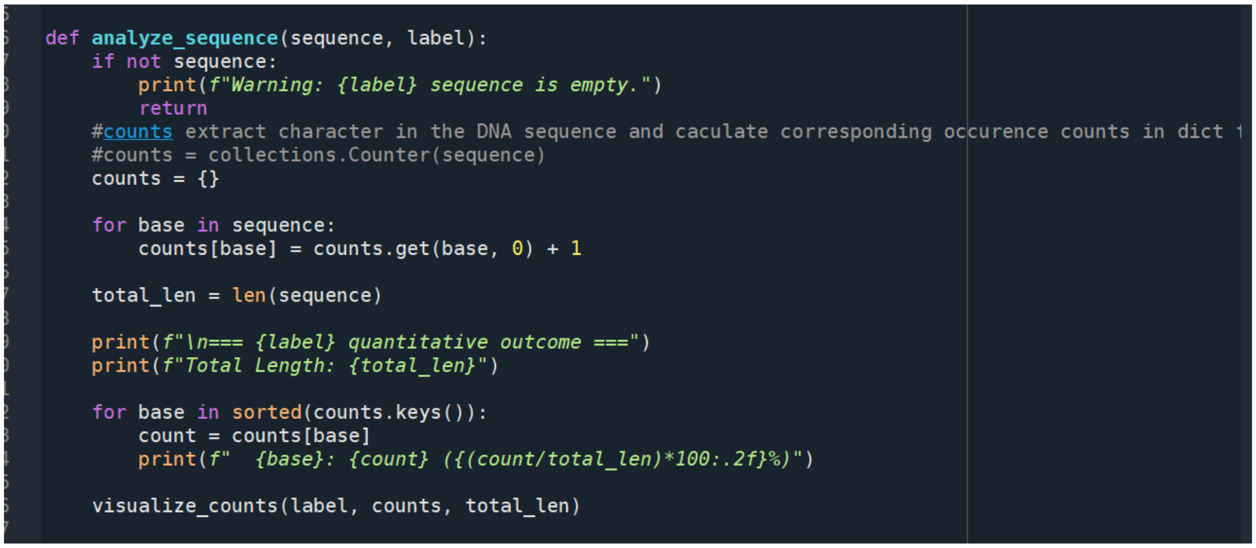
Second, I use strip('N') to remove the interfered 'N' characters from the start and end of the sequence.

Finally, I use join() method to merge the list into a single clean string.



```python
def read_file_sequence(file_path):
    sequence = []
    try:
        with open(file_path, 'r') as file:
            for line in file:
                line = line.strip() #remove the newline characters from each line
                if line:
                    sequence.append(line)
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
        return None

    return "".join(sequence).strip('N')
```

Step 2: Counting and Getting the frequencies of appearing base (analyze_sequence)



```python
def analyze_sequence(sequence, label):
    if not sequence:
        print(f"Warning: {label} sequence is empty.")
        return
    #counts extract character in the DNA sequence and caculate corresponding occurence counts in dict t
    #counts = collections.Counter(sequence)
    counts = {}

    for base in sequence:
        counts[base] = counts.get(base, 0) + 1

    total_len = len(sequence)

    print(f"\n=== {label} quantitative outcome ===")
    print(f"Total Length: {total_len}")

    for base in sorted(counts.keys()):
        count = counts[base]
        print(f"  {base}: {count} ({(count/total_len)*100:.2f}%)")

    visualize_counts(label, counts, total_len)
```

Goal: Getting the number of all kinds of bases and calculating their appearing frequencies.

Implementation: For every base encountered, I use the get() method to check if it already exists in the dictionary. If it does, I increment its count; If not (meaning a new base has appeared), I initialize it with a default value of 0 before incrementing. Finally, I use counts.values() divided by the total length of the sequence to represent the corresponding appeared frequency.

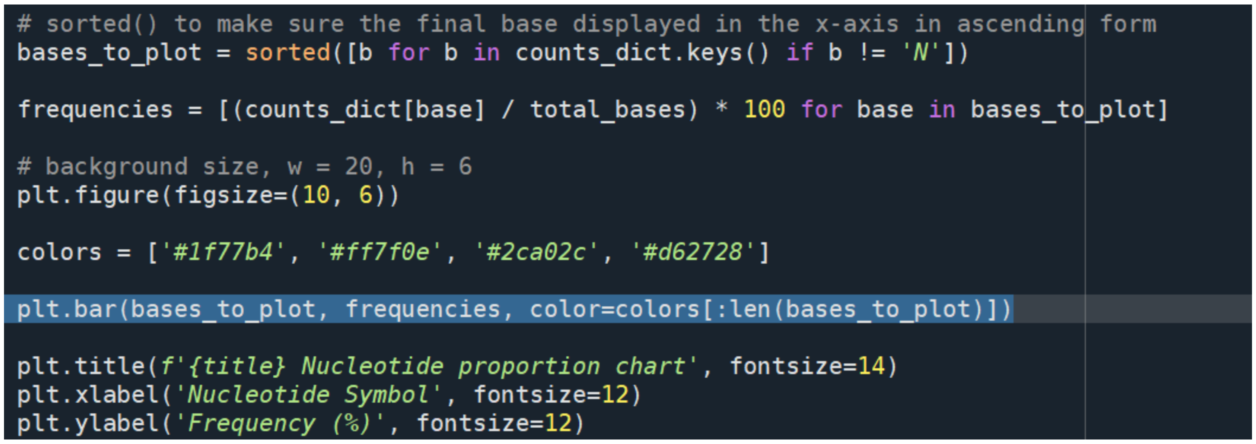
Step 3: Visualization (visualize_counts)

Implementation:

This function base_to_slot to sort the Martian bases in ascending order and plt.figure(figsize=(10,6)) to define the background canvas.

To display frequencies clearly, I use enumerate() to generate an index i. I then pass this index to plt.text() to position the labels. Specifically, I set the vertical postion to freq + 0.5 (placing the value slightly above the bar) and round the values to decimal places for readability.

Grid lines are drawn for each bar, and y-axis bounds are set.



```python
# sorted() to make sure the final base displayed in the x-axis in ascending form
bases_to_plot = sorted([b for b in counts_dict.keys() if b != 'N'])

frequencies = [(counts_dict[base] / total_bases) * 100 for base in bases_to_plot]

# background size, w = 20, h = 6
plt.figure(figsize=(10, 6))

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

plt.bar(bases_to_plot, frequencies, color=colors[:len(bases_to_plot)])

plt.title(f'{title} Nucleotide proportion chart', fontsize=14)
plt.xlabel('Nucleotide Symbol', fontsize=12)
plt.ylabel('Frequency (%)', fontsize=12)
```

2.1.3 Results

1. DNA Composition:

Identified Nucleotides: The nucleotides present in the Martian DNA sequence are A, C, T, and B.

Proportion:

- B: 29.41%
- C: 27.45%
- A: 22.06%

- T: 21.08%

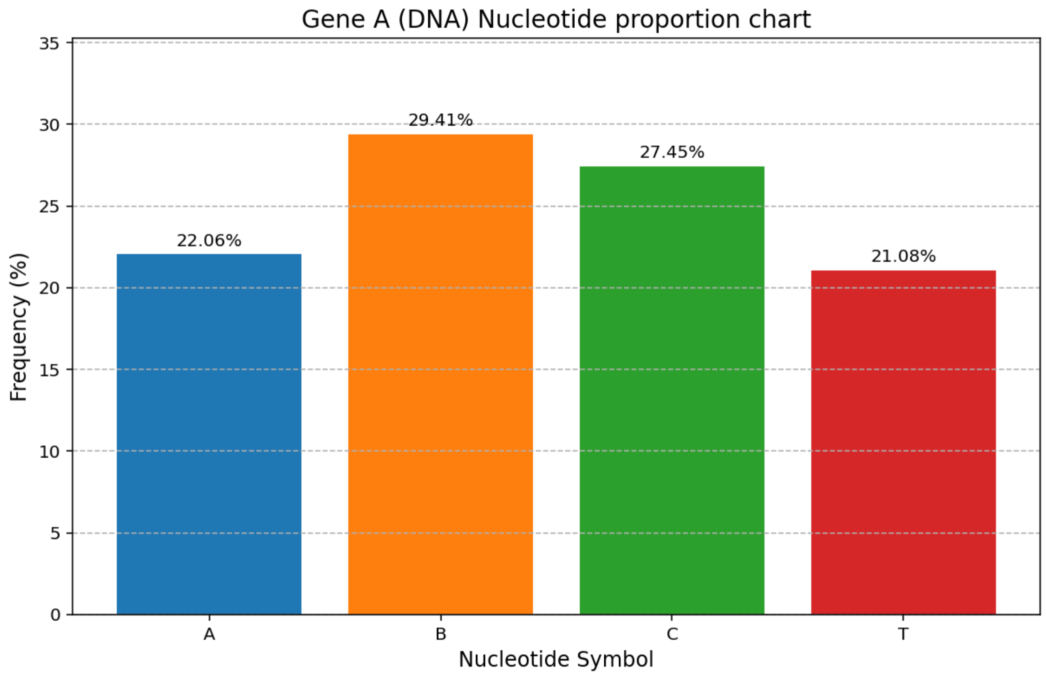


Figure 1: Nucleotide frequency distribution in Gene A (DNA)

2. RNA (mRNA) Composition:

Identified Nucleotides: The nucleotides present in the Martian RNA sequence are U, X, Y, and Z.

Proportion:

- Y: 29.41%
- X: 27.45%
- Z: 22.06%
- U: 21.08%

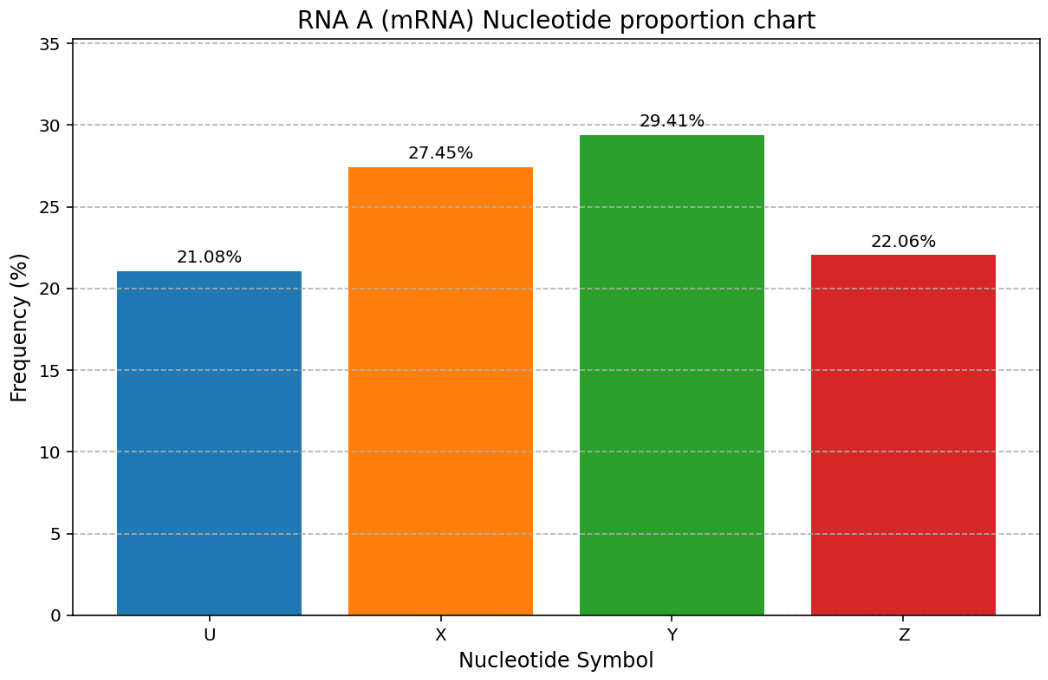


Figure 2: Nucleotide frequency distribution in RNA A (mRNA)

2.2 Task2 ——Discovering Base-Pairing Rules

2.2.1 Problem Analysis

1.The one-to-one pairing challenge and solution:

The principle of base complementary pairing is that when DNA sequence is transcribed into RNA sequence, each base in DNA sequence pairs exclusively with another base in the RNA sequence, no other possibilities allowed.

Therefore, I think the most challenging part of task2 lies in how to determine the unique pairing relationship between the DNA and RNA. After exploring several mathematical approaches, I have identified the final solution as follows:

I intend to divide the number of pairings between a base and each of its paired bases by the total number of pairings involving all the bases paired with the target base. If the ratio is not equal to one, it indicates one base is not 100% paired with another base, meaning this pairing outcome is wrong.

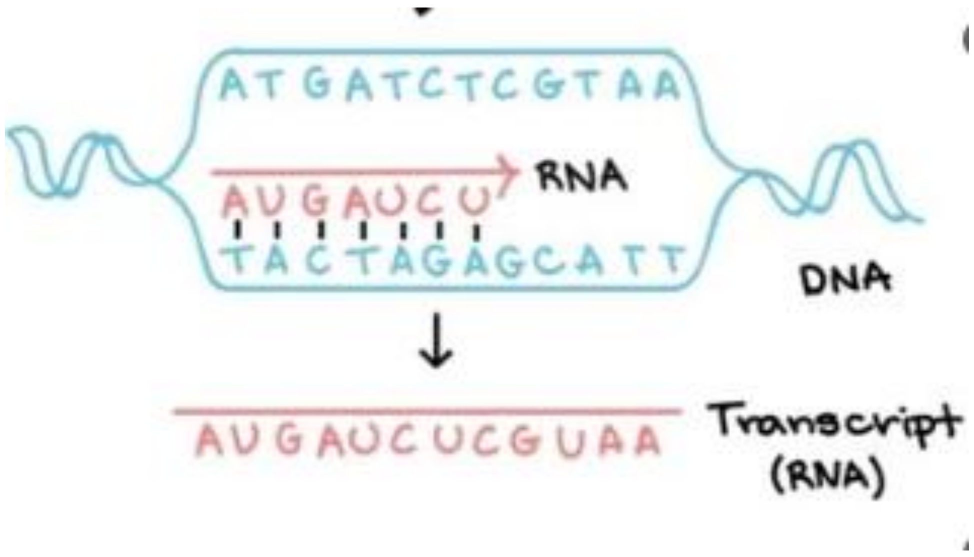


Figure 3: Illustration of the Central Dogma - Transcription process (Source: microbe notes).

2.2.2 Coding Implementation: Rule Derivation

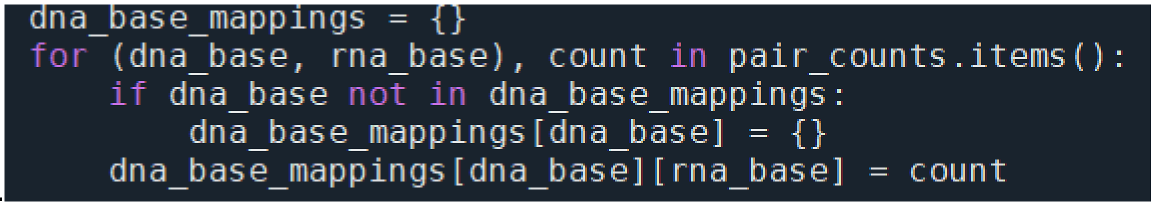
Managing Complex Data Relationships 's challenge and solution:

During the coding process, a significant difficulty was managing complex relationships between sequences. I found that using simple variables made it easy to get confused about which base mapped to which.

To solve this, I designed a specific nested dictionary structure to obtain data.

Implementation:

1.DNA to RNA nested dictionary:

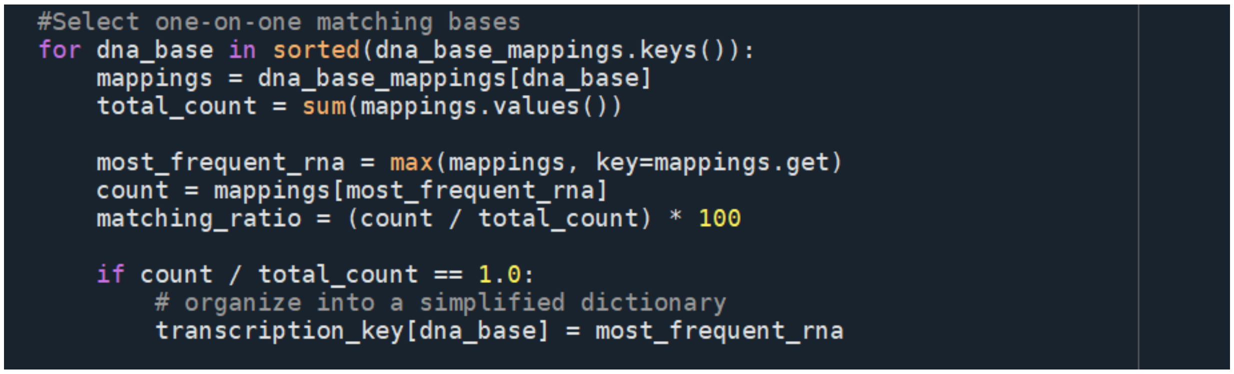


```
dna_base_mappings = {}
for (dna_base, rna_base), count in pair_counts.items():
    if dna_base not in dna_base_mappings:
        dna_base_mappings[dna_base] = {}
    dna_base_mappings[dna_base][rna_base] = count
```
.

I created a nested dictionary in this structure dna_base_mappings -> {dna_base : {rna_base : count}} by the coding block.

This rna_base and this count make up mappings in the structure of {rna_base : count} by the "mappings = dna_base_mappings[dna_base]".

Code flow:



```
#Select one-on-one matching bases
for dna_base in sorted(dna_base_mappings.keys()):
    mappings = dna_base_mappings[dna_base]
    total_count = sum(mappings.values())

    most_frequent_rna = max(mappings, key=mappings.get)
    count = mappings[most_frequent_rna]
    matching_ratio = (count / total_count) * 100

    if count / total_count == 1.0:
        # organize into a simplified dictionary
        transcription_key[dna_base] = most_frequent_rna
```

1.I use the max() method to identify the most frequent RNA partner. The coding sentence is: most_frequent_rna = max(mappings, key=mappings.get).

2.From this, I obtain the count of the occurrence. I then calculate the matching ratio by calculating count parameter divided by total_count.

3. use count / total_count == 1.0 to filter the only paired base. If the pairing outcome is not equal to one, representing the DNA is not unique paired with transcript RNA.

4. Final DNA to RNA pairing relationship: transciption_key -> {dna_base : most_frequent_rna}

2.Reverse Complementary Pairing (RNA to DNA)

Biological Principle: In bioinformatics, the relationship must be one to one. It is invalid for multiple RNA bases to point to the same DNA bases (e.g., A -> M and B -> M is impossible, where A, B are RNA and M is DNA).

2.2.3 Get aligned between the DNA sequence segment and RNA sequence

A Problem:

Given the simplicity of this question, the length of DNA is equal to the length of RNA after remove the ‘N’ from the DNA sequence, but in the real-world scenarios, an DNA sequence is much longer than the RNA sequence, the RNA sequence only paired with a small segment of DNA, this make us need to find the starting point of alignment. This seemed contradictory.

The Breakthrough:

I found the solution by using the universal T(Thymine) pair with U(Uracil) rule. We fix the DNA sequence in place and slide the RNA sequence along it. The sliding stops when every 'T' in the DNA pairs with a corresponding 'U' in the RNA. After that, we can identify the base pairs for the remaining nucleotides.

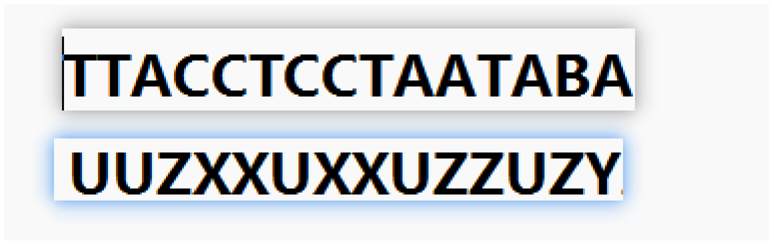
Process of Sliding Comparison:

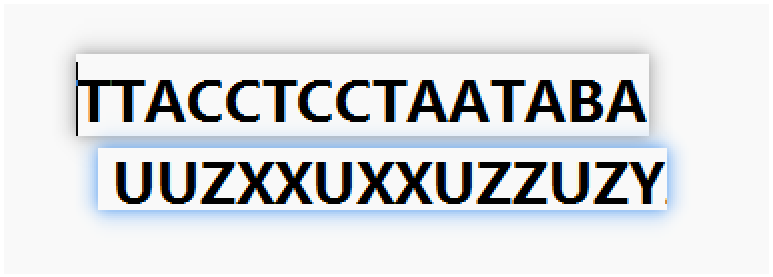
1. The first step is to find the started comparing point in DNA sequence, called offset. The scope of comparing is from 0 to dna_len – rna_len + 1(“+1” because of range function to determine the right boundary), because beyond that, the RNA sequence would exceed the DNA boundary.

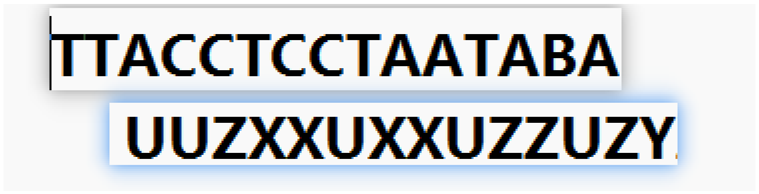
2.I use a max_matches parameter. In the loop, I extract the aligned position d from DNA and r from RNA. If d == ‘T’ and r == ‘U’, I add 1 to current_matches.

3.After comparison, max_matches represents the position where all T and U bases are paired. The best_offset is recorded as the correct matching position.

4.Finally, the get_aligned_sequences function uses this offset. aligned_dna gets the pairing DNA segment, and pair_counts mapping gets the matching base pair tuples. These are returned to the derive_transcription_rules function.







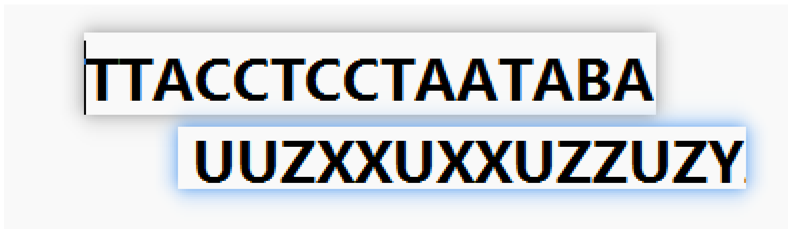
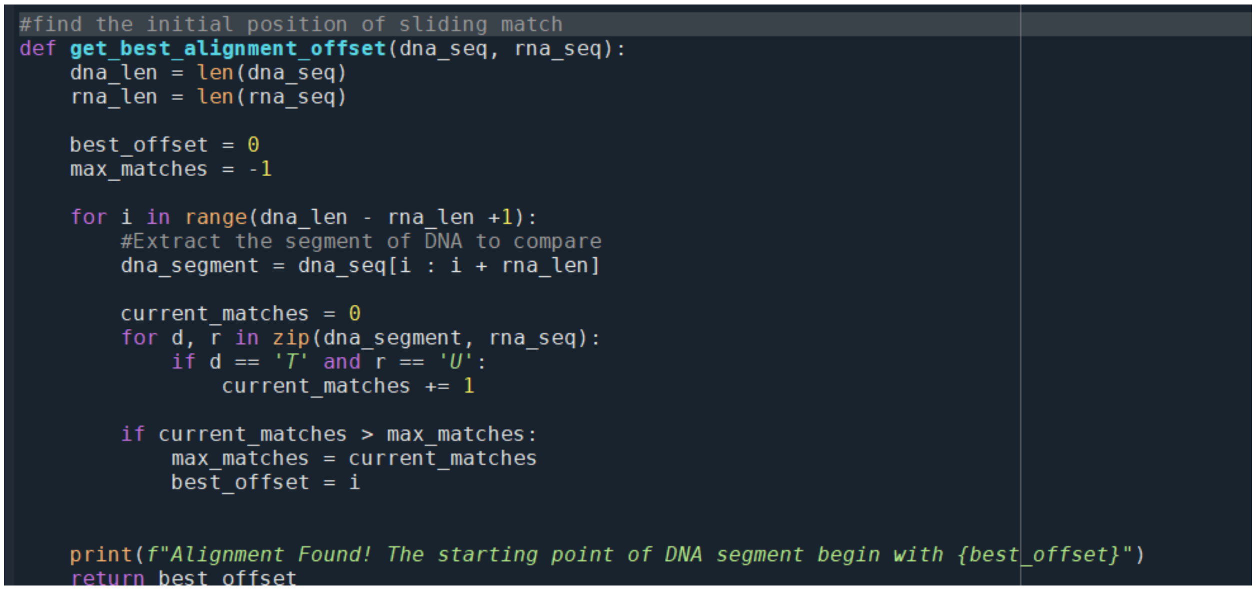


Figure 4:Visualization of the sliding window alignment algorithm

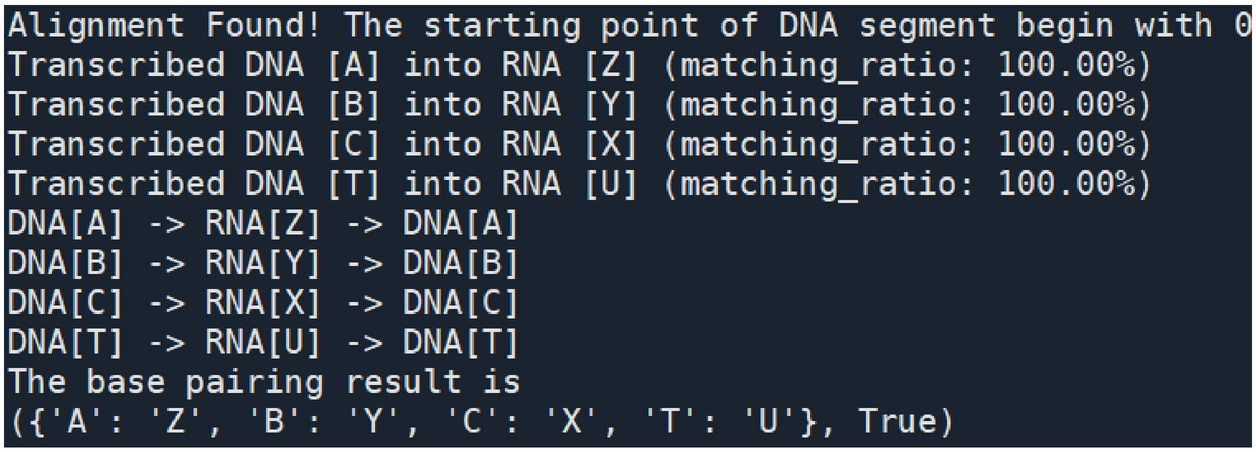


```python
#find the initial position of sliding match
def get_best_alignment_offset(dna_seq, rna_seq):
    dna_len = len(dna_seq)
    rna_len = len(rna_seq)

    best_offset = 0
    max_matches = -1

    for i in range(dna_len - rna_len +1):
        #Extract the segment of DNA to compare
        dna_segment = dna_seq[i : i + rna_len]

        current_matches = 0
        for d, r in zip(dna_segment, rna_seq):
            if d == 'T' and r == 'U':
                current_matches += 1

        if current_matches > max_matches:
            max_matches = current_matches
            best_offset = i


    print(f"Alignment Found! The starting point of DNA segment begin with {best_offset}")
    return best_offset
```

2.2.4 Results of Base Pairing

Based on the code execution, the unique pairing rules derived are:

DNA Base	RNA Base
A	Z
B	Y
C	X
T	U



```
Alignment Found! The starting point of DNA segment begin with 0
Transcribed DNA [A] into RNA [Z] (matching_ratio: 100.00%)
Transcribed DNA [B] into RNA [Y] (matching_ratio: 100.00%)
Transcribed DNA [C] into RNA [X] (matching_ratio: 100.00%)
Transcribed DNA [T] into RNA [U] (matching_ratio: 100.00%)
DNA[A] -> RNA[Z] -> DNA[A]
DNA[B] -> RNA[Y] -> DNA[B]
DNA[C] -> RNA[X] -> DNA[C]
DNA[T] -> RNA[U] -> DNA[T]
The base pairing result is
({'A': 'Z', 'B': 'Y', 'C': 'X', 'T': 'U'}, True)
```

This is the code execution result. A is paired with Z, B is paired with Y, C is paired with X, T is paired with U.

2.3 Task3: Codon Length Calculation

2.3.1 Coding Methodology and Coding Principle

Problem Solving Logic:

The primary objective of this task is to mathematically derive the codon length from the given mRNA and protein sequences. Since we do not know the rules of Martian biology beforehand, we need to test two possibilities regarding the Stop Codon:

1.Posibility A: The sequence contains a stop codon.

2.Posibility B: The sequence does not contain a stop codon.

$$Codon\ Length = \frac{Total\ Length\ of\ mRNA}{Total\ Number\ of\ Amino\ Acids}$$

$$Codon\ Length = \frac{Total\ Length\ of\ mRNA}{Total\ Number\ of\ Amino\ Acids + 1}$$

The coding logic is to substitute the values into the formula for both scenarios. If the calculation result is an integer, that scenario is the correct biological rule. As shown in the figure below, this “+1” represents the part that determines whether a stop codon is added.

2.3.2 Processing Protein sequence

Observation protein sequence

One error-prone issue here is that after examination of protein sequences, it is found that each amino acid is represented by a single elemental symbol (e.g., two letters like ‘Na’ or ‘Mg’) in Mar.

2.3.3 Code Implementation

Function Logic (get_codon_length):

This function receives rna_seq and protein_seq as inputs and executes the following flow:

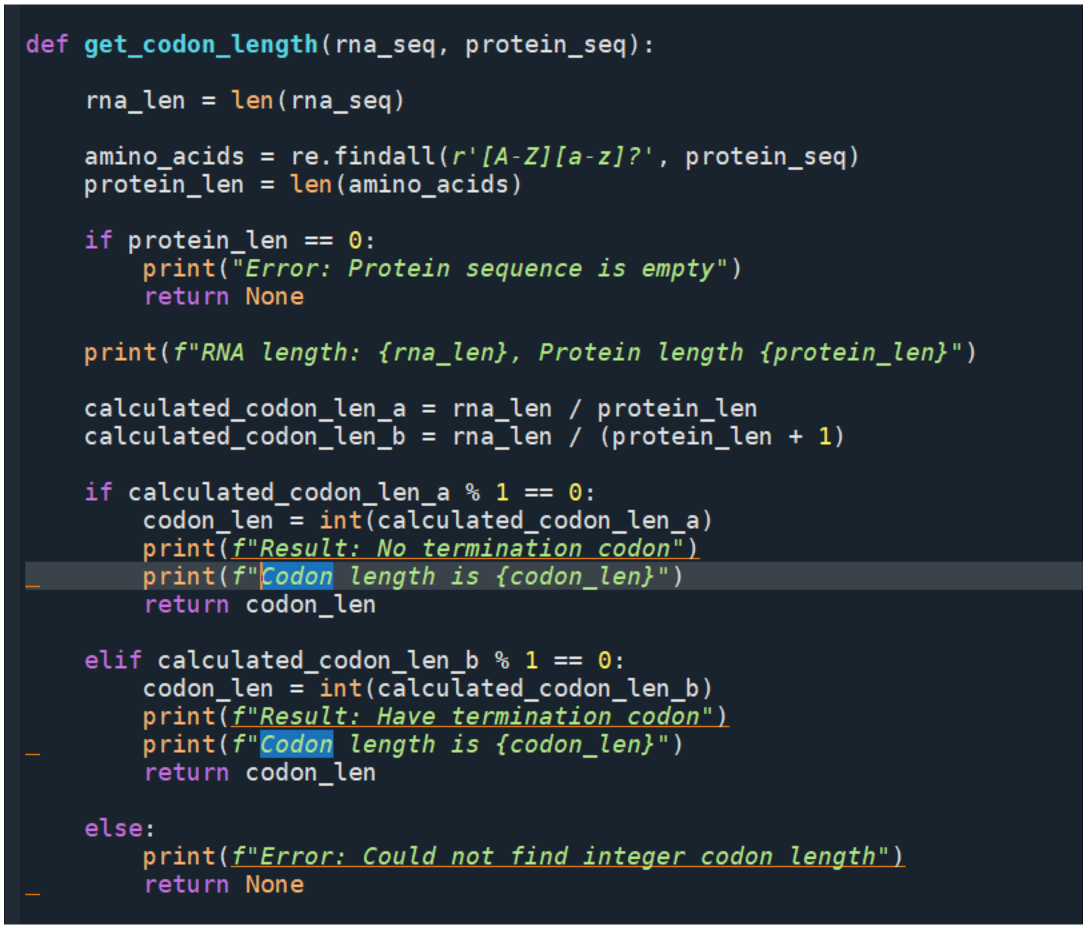
1.Data Processing: As described above, use regular expressions to convert the protein string into a list and calculate the process protein_len. This regular expression write like that “re.findall(r'[A-Z][a-z]?', protein_seq)”, because of the amino acid composed of one capital letter or one capital letter and a lowercase letter.

2.Judge if it exists a stop codon:

- calculated_codon_len_a = rna_len / protein_len
- calculated_codon_len_b = rna_len / (protein_len + 1)

3. Integer Validation:

I use the modulo operator calculated_codon_len % 1 == 0 to confirm whether the result is a valid integer.



```python
def get_codon_length(rna_seq, protein_seq):

    rna_len = len(rna_seq)

    amino_acids = re.findall(r'[A-Z][a-z]?', protein_seq)
    protein_len = len(amino_acids)

    if protein_len == 0:
        print("Error: Protein sequence is empty")
        return None

    print(f"RNA length: {rna_len}, Protein length {protein_len}")

    calculated_codon_len_a = rna_len / protein_len
    calculated_codon_len_b = rna_len / (protein_len + 1)

    if calculated_codon_len_a % 1 == 0:
        codon_len = int(calculated_codon_len_a)
        print(f"Result: No termination codon")
        print(f"Codon length is {codon_len}")
        return codon_len

    elif calculated_codon_len_b % 1 == 0:
        codon_len = int(calculated_codon_len_b)
        print(f"Result: Have termination codon")
        print(f"Codon length is {codon_len}")
        return codon_len

    else:
        print(f"Error: Could not find integer codon length")
        return None
```

2.3.4 Results

No termination of codon on the protein sequence.

Total RNA length	Total Protein Count	Calculated Codon Length
204	102	2

2.4 Task4: Construction of Amino Acid Lookup Table

2.4.1 Task requirements and biological principles

Coding Flow

1.Loop: I use the len(amnino_acids) as the number of loop iterations.

2.Positional Calculation: Inside the loop, I use a zero-based index i to calculate the specific position of each codon within the RNA string using the formula:

3.Extraction: I create the lookup table that codon corresponding to the current amino acid by slicing the RNA sequence : rna_seq[start : end].The structure is codon_table = {codon : curreent_aa}, that is codon_table = { rna_seq [start_index : end_index] : amino_acids}.

4.Implementation Details and Error Prevention

Boundary Safety Check:

A critical part of the implementation is preventing index out-of-bounds errors. I included a conditional judgment to check if end_index <= len(rna_seq).

This ensures that we only attempt to read the RNA sequence if a full codon exists.

If valid, I store the correspondence in the dictionary codon_table. I also check if codon not in codon_table before assigning, which means creating newly identified bases.

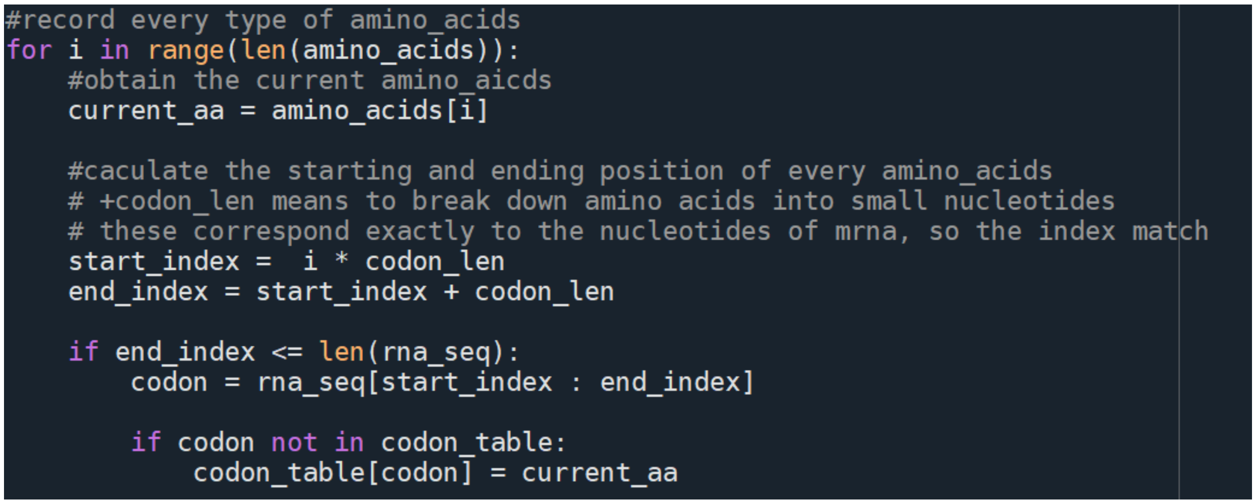
Stop Codon Detection Logic:

Although I know about the Gene a does not exist stop codon, for preventing another gene exists stop codons, so I keep creating stop codon correspondence in the lookup table.

I implemented a specific logic to identify stop codons based on sequence length comparison.

I calculated the expected_rna_len (which is the Number of Amino Acids × Codon Length).

Logic : If the actual len(rna_seq) is longer than expected_rna_len, it implies that there is a segment of the RNA sequence that is not translated into amino acids. This remaining segment is identified as the stop codon.



```
#record every type of amino_acids
for i in range(len(amino_acids)):
    #obtain the current amino_aicds
    current_aa = amino_acids[i]

    #caculate the starting and ending position of every amino_acids
    # +codon_len means to break down amino acids into small nucleotides
    # these correspond exactly to the nucleotides of mrna, so the index match
    start_index =  i * codon_len
    end_index = start_index + codon_len

    if end_index <= len(rna_seq):
        codon = rna_seq[start_index : end_index]

        if codon not in codon_table:
            codon_table[codon] = current_aa
```

2.4.3 Results

Using the derived codon length, the lookup_table_mrna_to_amino function successfully generated a translation map.

The resulting codon table demonstrates a strict one-to-one mapping between codons and amino acids based on protein_a and rna_a dataset.

MRNA codon	Amino Acid
UU	As
ZX	Be
XU	N
XX	S
UZ	Mg
ZU	Ni
ZY	Li
YY	Pb
YU	Cl
UX	K
YX	Rn
XY	Al
UY	Na
XZ	C
YZ	Ar
ZZ	Zn

2.5 Task5: Generating Gene b’s mRNA and amino acid sequences

2.5.1 Problem

Due to gene b haven’t rna and protein sequence information, we need to learn basic rules of transcription and translation from gene a.

2.5.2 The Coding function

generate_basic_information() function:

I designed a caller function named generate_basic_information function used for it. It invokes functionalities from other modules to obtain the necessary rules. It contains 4 parts to derive transcription key, codon length and codon-to-amino acids lookup table. It performs four steps:

1. Read Gene b’s sequences.

2. Get a transcription key.

3. Get codon length.

4.Get codon-to-amino acid lookup table.

transcribe_dna_to_rna() function:

It is used to obtaining the mRna sequence from the DNA sequence.

translate_rna_to_protein() function:

It is used for getting protein sequences. If there are any codons that do not appear in the Gene a sequence before, it will use ‘X’ to replace the amino acids, which means the codon in the gene b hasn’t appeared in Gene a before.

main() function:
It is mainly used for generating Gene b’s amino acid and mRNA sequences.

2.5.3 Results

Gene b’s mRNA sequence:
UUYZXYUXUYYUXUZXZXYXYUUYYZUXZUYYZXXXYXUYYZYYUZZYZUZYYZYUXYUZYXY
ZXZXUYUYYXUUXXYXUZYUYYUXXUYXXYXXXUYUXZXZYYYUXUZXUXZYZUZUXYXXYUY
XYXUXUYUXZUXZUYXXUXXXYZYYUUZXYYYZUXUXYUZUYUXXZXZYYUYYZXXXYYZXUU
YXZXZUXYZUYYUZZ

Gene b’s protein sequence:

AsArAlKNaClNBeBeRnClNaArKNiPbBeSRnNaArPbMgLiNiLiArClAlMgRnArCNClPbNKAlNLiNa
ClSNaSRnSNaKBeLiPbKMgNCArMgKRnAlNaAlNNClCKNiRnNSAlLiClMgAlPbNiNAlMgNaKCC
PbNaArSAlArNNaCCKArNaClZn

3.Disscusion

3.1 Comparison of Genetic Composition

1. DNA Comparison:
On Earth, the standard DNA nucleotides are A, T, C, and G. In contrast, the Martian DNA consists of A, C, T, and B. Differences: We can clearly conclude that B is not an Earth nucleotide, and we have not detected the presence of G (Guanine) in the Martian samples. The genetic alphabet is fundamentally different.

2. RNA Comparison:
On Earth, the standard RNA nucleotides are A, U, C, and G. The Martian RNA consists of U, X, Y, and Z.

- Differences: The nucleotides X, Y, and Z do not exist in Earth's RNA structure.
- Similarities: The U (Uracil) nucleotide appears in both Earth and Martian RNA.

	Earth Code	Martian Code
DNA bases	A, T, C, G	A, T, C, B
RNA bases	A, U, C, G	U, X, Y, Z

3.2 Comparison of Base pairing

Similarities:

1. Both Earth and Martian genetic codes rely on one-to-one complementarity.

2. The pairing of DNA-T and RNA-U is the same on Earth and Martian.

Differences:

1. The transcription key is different.

Earth	Mars
A->U	A->Z
T->A	T->U
C->G	C->X
G->C	B->Y

3.3 Comparison of Codon length and possible combinations

The G(Guanine) is absent from the Martian DNA, and base B appears.

The use of Z, X, Y base instead of Earth's standard A, C, G.

Task3 Discussion: The impact of different Codon length

- Different number of possible codon combinations:

  Earth: $4^3=64$ possible combinations.

  Mars: $4^2=16$ possible combinations.

This makes Earth have higher degeneration (64 codons for 20 amino acids) than Mars (16 codons for 16).

	Earth Code	Martian Code
Codon length	3	2
Possible Codon Combinations	64	16
Degeneracy	High	Low

4.Conclusion

In this report, we successfully deciphered the genetic language of the Martian species by analyzing the Gene A dataset and applying the rules to the Gene B sequence. The project identified the unique base, transcription key, codon length, amino acid sequence, and codon-to-amino acids correspondent. And through these results, we discuss the similarities and differences between Mars and Earth. From that, we broaden our horizons to learn more about the Martian genetic code.

5. References

1. Microbe Notes (2018). Differences between DNA Replication and Transcription. [Online]. Available at: https://microbenotes.com/differences-between-dna-replication-and-transcription/ [Accessed: 14 December 2025].