



Coding for Biologists with Biopython

Course Code: XXXXXXXX

Syllabus and Manual

Updated: 11th feb 2025

Coding for Biologists with BioPython

Syllabus

Reference: <https://biopython.org/DIST/docs/tutorial/Tutorial.html>

Lab Exercises

Unit 1: Introduction to Biopython and Unit Sequencing Biopython: Chapters 1 to 5	
1.1	Setting Up the Environment and Introduction to Biopython • Install Biopython • Explore the Biopython documentation and tutorial • Write a simple Python script to print "Hello, Biopython!"
1.2	Sequence Objects • Create sequence objects using Seq • Perform basic sequence manipulations (slicing, concatenation) • Transcription and translation of sequences
1.3	Reading Sequence Files • Read sequence data from FASTA and GenBank files • Extract sequences and annotations
1.4	Writing Sequence Files • Write sequence data to FASTA and GenBank files • Convert between different file formats
Unit 2: Sequence annotations, alignments and Database Access Biopython: Chapters 4 to 9	
2.1	Sequence Annotation Objects • Explore SeqRecord objects • Add and manipulate annotations (features, IDs, descriptions)
2.2	Accessing Online Databases • Use Entrez to fetch data from NCBI • Retrieve nucleotide and protein sequences • Parse XML data from Entrez
1.6	Pairwise Sequence Alignment • Perform pairwise sequence alignment using Bio.pairwise2 • Score and visualize alignments
	Multiple Sequence Alignment • Use Clustalw or MUSCLE for multiple sequence alignments • Analyze and visualize the alignment results
Unit 3: Phylogenetics and Population Genetics	

Biopython: Chapters 7 to 8	
2.1	Constructing Phylogenetic Trees • Use alignment data to construct phylogenetic trees • Visualize phylogenetic trees using Phylo
2.2	Population Genetics Analysis • Simulate population genetics data using Bio.PopGen • Analyze genetic variation and structure in populations
Unit 4: Protein Structures and Pipeline Biopython: Chapters 10 and 11	
2.2	Working with Protein Structures • Fetch protein structure data from PDB • Visualize protein structures using Bio.PDB • Perform basic structure manipulations
	Building a Bioinformatics Pipeline • Combine multiple Biopython modules to build a complete bioinformatics pipeline • Perform a real-world biological data analysis
Unit 5: Final Project Biopython: Chapters 1 to 11	
5.1	Project 1: Comparative Genomics and Phylogenetic Analysis • Analyze evolutionary relationships between related species • Perform multiple sequence alignment and construct a phylogenetic tree
5.2	Project 2: Protein Structure Analysis and Functional Prediction • Analyze the structure of a protein and predict its functional sites • Compare with related proteins to understand its biological role

1.	a. Describe the steps to install Biopython in your Python environment. b. Using the Seq class from Biopython, create a DNA sequence object and: i. Slice the sequence to extract a specific region (e.g., from index 3 to 10). ii. Concatenate this sequence with another sequence. iii. Transcribe and translate the concatenated sequence into RNA and protein sequences.
2.	a. What is the role of the Seq class in Biopython, and how do operations like slicing, concatenation, transcription, and translation help in bioinformatics when working with DNA, RNA, and protein sequences? b. Write a Python script that reads sequence data from a FASTA file and extracts both the sequence and its description (header), then prints them.
3.	a. What is the GenBank file format, and how does the SeqRecord object in Biopython store DNA sequences and their annotations for export? b. Using a SeqRecord object, write the DNA sequence along with its annotations (e.g., gene name, function) to a GenBank file format.
4.	a. What is the difference between FASTA and GenBank file formats, and how can you convert sequence data from a FASTA file into the GenBank format while preserving both the sequence and its annotations? b. Given a FASTA file, write a Python script that reads the file and converts it into GenBank format, while preserving the sequence and annotations.
5.	a. What are SeqRecord objects in Biopython, and how can you use them to store DNA sequences along with annotations such as gene start and end positions, descriptions, and other features? How can annotations be added, modified, and manipulated in a SeqRecord object? b. Create a SeqRecord object for a DNA sequence and add annotations for a gene (start, end position, description). Modify the annotations and print the updated SeqRecord.
6.	a. What is Entrez, and how can it be used to fetch nucleotide sequences and metadata from NCBI databases? b. Write a script that uses Entrez to fetch a nucleotide sequence from the NCBI database by using a known accession number, and print out the sequence and the related metadata.
7.	a. What is pairwise sequence alignment, and why is it important in bioinformatics? b. Using Bio.pairwise2, perform pairwise sequence alignment of two DNA sequences. Print the alignment result and the alignment score.
8.	a. What is multiple sequence alignment (MSA), and why is it used in bioinformatics? b. Write a Python script that takes a list of protein sequences in FASTA format and performs a multiple sequence alignment using MUSCLE. Display the alignment result.
9.	a. What is a phylogenetic tree, and what is its role in bioinformatics? b. Using alignment data, construct a phylogenetic tree and visualize it with Bio.Phylo. Label each branch with the sequence name.
10.	a. What is the Protein Data Bank (PDB), and how is it used to access 3D protein structure data?

	b. Fetch a protein structure from the Protein Data Bank (PDB) using Bio.PDB and visualize the 3D structure of the protein. Perform basic manipulations like selecting a region or displaying specific chains.
--	---

1. a. Describe the steps to install Biopython in your Python environment.

Answer: To install Biopython in your Python environment, follow these steps:

1. Ensure Python and pip are installed:

First, make sure that Python and pip (Python's package installer) are installed on your system. You can check if Python is installed by running:

python --version

Or, for some systems, you might need to use:

python3 --version

Similarly, check if pip is installed by running:

pip --version

If these are not installed, you'll need to install Python and pip first.

2. Install Biopython:

Once you have Python and pip installed, you can install Biopython using pip. Run the following command in your terminal or command prompt:

pip install biopython

Or, if you are using Python 3 and pip is associated with Python 2:

pip3 install biopython

3. Verify Installation:

To verify that Biopython has been installed successfully, open a Python interpreter (by running python or python3 in your terminal) and try importing Biopython:

from Bio.Seq import Seq

If no errors occur, Biopython is correctly installed.

1 b. Using the Seq class from Biopython, create a DNA sequence object and: i. Slice the sequence to extract a specific region (e.g., from index 3 to 10). ii. Concatenate this sequence with another sequence. iii. Transcribe and translate the concatenated sequence into RNA and protein sequences.

```
from Bio.Seq import Seq

# Step 1: Create a DNA sequence object
dna_sequence = Seq("ATGCTAGCTAGCTAGCTG")

# Step 2: Slice the sequence from index 3 to 10
sliced_sequence = dna_sequence[3:11]
print("Sliced Sequence:", sliced_sequence)

# Step 3: Concatenate with another sequence
another_sequence = Seq("GGCTAG")
concatenated_sequence = sliced_sequence + another_sequence
print("Concatenated Sequence:", concatenated_sequence)

# Step 4: Transcribe the concatenated sequence into RNA
rna_sequence = concatenated_sequence.transcribe()
print("RNA Sequence:", rna_sequence)

# Step 5: Translate the RNA sequence into a protein sequence
protein_sequence = rna_sequence.translate()
print("Protein Sequence:", protein_sequence)
```

Expected Output:

Sliced Sequence: GCTAGCT

Concatenated Sequence: GCTAGCTGGCTAG

RNA Sequence: GCUAGCUGGCUAG

(if warning is encounter because RNA sequence is not a multiple of 3)

Protein Sequence: ALGV*

2 a. What is the role of the Seq class in Biopython, and how do operations like slicing, concatenation, transcription, and translation help in bioinformatics when working with DNA, RNA, and protein sequences?

Answer: DNA, RNA, and protein sequences are fundamental elements in molecular biology:

- **DNA sequence:** This is a long chain of nucleotides that contain the genetic instructions for the development and functioning of living organisms. It is composed of four bases: adenine (A), thymine (T), cytosine (C), and guanine (G).
- **RNA sequence:** This is a single-stranded molecule that plays a central role in the translation of genetic information from DNA into proteins. It uses uracil (U) instead of thymine (T).
- **Protein sequence:** This is a chain of amino acids linked together, and it is encoded by RNA through a process called translation. Proteins perform essential functions within cells.

In Biopython, the Seq class is used to represent and manipulate DNA, RNA, and protein sequences. It provides convenient methods for performing various operations:

1. **Slicing:** You can extract a specific region of a sequence using slice notation (e.g., from index 3 to 10). This allows you to focus on a particular subsequence of interest, such as a gene or regulatory region.
2. **Concatenation:** Sequences can be joined together using the + operator, which allows you to combine multiple sequences (e.g., merging a gene with its regulatory elements).
3. **Transcription:** This process involves converting a DNA sequence into RNA. In Biopython, the transcribe() method of the Seq class allows you to perform this operation by replacing thymine (T) with uracil (U).
4. **Translation:** The process of converting an RNA sequence into a protein sequence is known as translation. The translate() method in Biopython converts an RNA sequence into the corresponding protein sequence by mapping codons to amino acids.

These operations are crucial in bioinformatics for analyzing and interpreting biological sequences, as they allow the manipulation of genetic information at different levels—DNA, RNA, and protein.

b. Python Script to Read Sequence Data from a FASTA File and Extract Both the Sequence and its Description (Header)

```
from Bio import SeqIO

# Function to read sequences from a FASTA file and print description and sequence
def read_fasta(file_path):
    # Parse the FASTA file
    for record in SeqIO.parse(file_path, "fasta"):
        # Print the description (header) and sequence
        print(f"Description: {record.description}")
        print(f"Sequence: {record.seq}")
        print() # Print a blank line between records

# Specify the path to your FASTA file
fasta_file = "example.fasta" # Replace with your actual file path

# Call the function to read and print the sequence data
read_fasta(fasta_file)
```

Example FASTA file (example.fasta):

```
>seq1 This is the description for sequence 1
ATGCATGCGTACGTAGCTA
>seq2 This is the description for sequence 2
GCTAGCTAGCTAGCTA
```

Expected Output:

```
Description: seq1 This is the description for sequence 1
Sequence: ATGCATGCGTACGTAGCTA
Description: seq2 This is the description for sequence 2
Sequence: GCTAGCTAGCTAGCTA
```

3 a. What is the GenBank file format, and how does the SeqRecord object in Biopython store DNA sequences and their annotations for export?

The **GenBank file format** is a widely used format for storing nucleotide sequence data along with its associated annotations. It contains information such as sequence features, gene names, product functions, and more. A GenBank file typically consists of two main sections:

1. **Sequence Data:** The nucleotide or protein sequence.
2. **Annotations:** Metadata such as gene names, protein functions, and locations of important features (e.g., exons, regulatory regions).

The **SeqRecord object** in Biopython is designed to hold both the **sequence** and its **annotations** in a structured way. It stores the sequence as a Seq object and allows you to attach additional information, such as:

- **Annotations:** Metadata like gene names, function descriptions, and other biological information.
- **Features:** Specific regions of the sequence (e.g., coding regions, exons).
- **ID, Name, and Description:** Useful for identifying the sequence in a broader database or file.

Using the SeqRecord object, you can export DNA sequences along with their annotations to the GenBank file format using Biopython's SeqIO.write() method.

b. Using a SeqRecord object, write the DNA sequence along with its annotations (e.g., gene name, function) to a GenBank file format

```
from Bio.Seq import Seq

from Bio.SeqRecord import SeqRecord

from Bio import SeqIO

# Step 1: Create a DNA sequence

dna_sequence = Seq("ATGCGTACGTAGCTAGCTAG")

# Step 2: Create a SeqRecord object with annotations

record = SeqRecord(

    dna_sequence,

    id="seq1",
```

```

name="Example_Gene",

description="Example gene sequence",

annotations={

    "molecule_type": "DNA", # Required for GenBank format

    "gene": "ExampleGene",

    "function": "Hypothetical protein"

}

)

```

Step 3: Write the SeqRecord object to a GenBank file

```
output_file_path = "C:/Users/Admin/Downloads/q4_genbank.gb"
```

```
with open(output_file_path, "w") as output_file:
```

```
    SeqIO.write(record, output_file, "genbank")
```

```
print("GenBank file written successfully.")
```

Step 4: Open and read the GenBank file

```
with open(output_file_path, "r") as input_file:
```

```
    record_read = SeqIO.read(input_file, "genbank")
```

```
    print("\nContents of the GenBank file:")
```

```
    print(record_read)
```

Example of the output in the GenBank file:

```

LOCUS seq1                21 bp DNA linear UNK 01-JAN-2025
DEFINITION Example gene sequence.
ACCESSION seq1
VERSION seq1.1
KEYWORDS .

```

```

SOURCE    Synthetic construct
ORGANISM  Synthetic construct
REFERENCE 1 (bases 1 to 21)
AUTHORS   Example Author
TITLE     Direct submission
FEATURES             Location/Qualifiers
     gene             1..21
                     /gene="ExampleGene"
                     /function="Hypothetical protein"
ORIGIN
     1 atcggtacgt agctagctag
//

```

4 a. What is the difference between FASTA and GenBank file formats, and how can you convert sequence data from a FASTA file into the GenBank format while preserving both the sequence and its annotations?

The **FASTA** and **GenBank** file formats are both commonly used for storing sequence data in bioinformatics, but they differ in terms of the information they contain:

1. **FASTA Format:**

- **Structure:** FASTA files store only the sequence data, along with a simple header line (starting with a '>' symbol) that typically contains a brief identifier or description of the sequence.
- **Content:** FASTA format includes the sequence itself (DNA, RNA, or protein), but it does **not** store detailed annotations like gene names, sequence features, or functional descriptions.

2. **GenBank Format:**

- **Structure:** GenBank files provide a much more detailed record that includes sequence data, annotations, and additional metadata such as gene names, product descriptions, sequence features (e.g., coding regions, exons), and publication references.
- **Content:** GenBank format stores not only the sequence but also features such as location of genes, coding regions, exons, and other functional or structural annotations. This makes GenBank more useful for storing biologically rich sequence data.

Conversion Process:

To convert sequence data from a **FASTA file** to **GenBank format**, the conversion process must:

- **Read the FASTA file** to extract the sequence and the description.
- **Create a SeqRecord object** in Biopython, which stores both the sequence and any annotations (even if they are minimal).
- **Write the SeqRecord object to a GenBank file**, where you can include sequence features (e.g., gene name, function, locations).

b. Given a FASTA file, write a Python script that reads the file and converts it into GenBank format, while preserving the sequence and annotations.

```

from Bio import SeqIO

from Bio.SeqRecord import SeqRecord

# Function to convert FASTA file to GenBank format

def convert_fasta_to_genbank(fasta_file, genbank_file):

    # Parse the FASTA file and read sequences

    records = []

    for record in SeqIO.parse(fasta_file, "fasta"):

        # Extract the sequence and description from FASTA record

        sequence = record.seq

        description = record.description

        # Create SeqRecord object for GenBank with basic annotations

        genbank_record = SeqRecord(

            sequence,

            id=record.id,

            name="Example_Gene",

            description=description,

            annotations={

                "molecule_type": "DNA", # Required for GenBank format

                "gene": "ExampleGene",

                "function": "Hypothetical protein"

            }

```

```

    )

    records.append(genbank_record) # Add the record to the list

# Write all SeqRecords to GenBank format at once

with open(genbank_file, "w") as output_handle:

    SeqIO.write(records, output_handle, "genbank")

print(f"All FASTA sequences converted to GenBank format and saved as {genbank_file}")

# Define input and output file paths

fasta_file = "C:/Users/Admin/Downloads/fasta_1.fasta" # Replace with your actual FASTA file path

genbank_file = "example_output.gb" # Output GenBank file

# Call the function to convert FASTA to GenBank

convert_fasta_to_genbank(fasta_file, genbank_file)

```

Example FASTA file (example.fasta):

```

>seq1 Example sequence description
ATGCGTACGTAGCTAGCTAG

```

Expected Output in the GenBank file (example_output.gb):

```

LOCUS seq1                21 bp DNA linear UNK 01-JAN-2025
DEFINITION Example sequence description
ACCESSION seq1
VERSION seq1.1
KEYWORDS .
SOURCE Synthetic construct
ORGANISM Synthetic construct
REFERENCE 1 (bases 1 to 21)
AUTHORS Example Author
TITLE Direct submission
FEATURES             Location/Qualifiers
     gene             1..21
                     /gene="ExampleGene"
                     /function="Hypothetical protein"
ORIGIN
      1 atcggtacgt agctagctag
//

```

5 a. What are SeqRecord objects in Biopython, and how can you use them to store DNA sequences along with annotations such as gene start and end positions, descriptions, and other features? How can annotations be added, modified, and manipulated in a SeqRecord object?

The **SeqRecord** object in **Biopython** is a key data structure that is used to represent sequence data (such as DNA, RNA, or protein sequences) along with associated metadata (annotations and features). It stores both the sequence itself and additional information about that sequence, which is essential for bioinformatics analyses.

A SeqRecord object consists of the following main components:

1. **Sequence (Seq):** This is the actual sequence data, which can be a DNA, RNA, or protein sequence.
2. **ID:** A unique identifier for the sequence.
3. **Name:** A simple name for the sequence (often used to refer to it in a simpler way).
4. **Description:** A short description or header that provides additional context about the sequence.
5. **Annotations:** A dictionary containing metadata about the sequence, such as gene names, functions, locations, references, and other relevant biological information.
6. **Features:** These are more specific locations or regions within the sequence (e.g., exons, coding regions), each of which can be annotated with attributes such as the type of feature (e.g., gene, CDS) and additional metadata.

Working with Annotations:

- **Adding Annotations:** You can add annotations by modifying the annotations dictionary in a SeqRecord. For example, you can add a gene name or a description of the sequence's function.
- **Modifying Annotations:** Once annotations are added, they can be modified directly by updating the corresponding dictionary entries.
- **Manipulating Features:** Features such as gene positions or functional regions can be added or modified in the features list of a SeqRecord object. Each feature can store attributes such as location, type (e.g., "gene", "CDS"), and qualifiers (e.g., gene name, function).

b. Create a SeqRecord object for a DNA sequence and add annotations for a gene (start, end position, description). Modify the annotations and print the updated SeqRecord.

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

# Step 1: Create a DNA sequence
dna_sequence = Seq("ATGCGTACGTAGCTAGCTAG")

# Step 2: Create a SeqRecord object with the sequence
record = SeqRecord(
    dna_sequence,
    id="seq1",
    name="Example_Gene",
    description="An example DNA sequence for gene annotation.",
)

# Step 3: Add annotations for the gene
record.annotations["gene"] = "ExampleGene"
```

```

record.annotations["function"] = "Hypothetical protein"
record.annotations["organism"] = "Synthetic organism"

# Step 4: Add a feature for the gene (start and end positions)
from Bio.SeqFeature import SeqFeature, FeatureLocation
gene_feature = SeqFeature(FeatureLocation(0, 21), type="gene", qualifiers={"gene": "ExampleGene"})
record.features.append(gene_feature)

# Step 5: Modify the annotation (change function description)
record.annotations["function"] = "Hypothetical protein with modified function"

# Step 6: Print the updated SeqRecord
print(f"ID: {record.id}")
print(f"Name: {record.name}")
print(f"Description: {record.description}")
print(f"Annotations: {record.annotations}")
print(f"Features: {record.features}")

```

Output:

ID: seq1

Name: Example_Gene

Description: An example DNA sequence for gene annotation.

Annotations: {'gene': 'ExampleGene', 'function': 'Hypothetical protein with modified function', 'organism': 'Synthetic organism'}

Features: [SeqFeature(FeatureLocation(ExactPosition(0), ExactPosition(21)), type='gene', qualifiers=...)]

6 a. What is Entrez, and how can it be used to fetch nucleotide sequences and metadata from NCBI databases?

Explanation: Entrez is a suite of tools developed by the National Center for Biotechnology Information (NCBI) that allows users to search and retrieve data from various biological databases, including nucleotide and protein sequences. Entrez provides programmatic access through the Entrez Programming Utilities (E-utilities), which can be used to query databases like GenBank and retrieve sequences, metadata, and other related information.

- Entrez is used for:
 - Searching and retrieving sequence data from the NCBI databases.
 - Fetching sequence metadata such as organism name, gene description, sequence length, etc.
 - Accessing biological literature and genome data.

In Python, the **Entrez** module from **Biopython** is used to interact with the NCBI Entrez system. You can fetch sequence data by providing an accession number and retrieve metadata such as the sequence's description, gene name, organism, and more.

b. Write a script that uses Entrez to fetch a nucleotide sequence from the NCBI database by using a known accession number, and print out the sequence and the related metadata.

```
from Bio import Entrez

# Step 1: Provide your email for NCBI's Entrez system
Entrez.email = "your_email@example.com"

# Step 2: Specify the accession number of the sequence
accession_number = "NM_001301717" # Example accession number for a human gene

# Step 3: Fetch the sequence from GenBank using Entrez
handle = Entrez.efetch(db="nucleotide", id=accession_number, rettype="gb", retmode="text")
record = handle.read()

# Step 4: Parse the sequence and metadata
from Bio import SeqIO

# Use SeqIO to parse the GenBank format sequence data
handle.seek(0)
seq_record = SeqIO.read(handle, "genbank")

# Step 5: Print the sequence and metadata
print(f"Accession Number: {seq_record.id}")
print(f>Description: {seq_record.description}")
print(f"Organism: {seq_record.annotations['organism']}")
print(f"Sequence: {seq_record.seq}")
print(f"Length of Sequence: {len(seq_record.seq)}")
print(f"Features: {seq_record.features}")
```

7 a. What is pairwise sequence alignment, and why is it important in bioinformatics?

Answer: Pairwise sequence alignment is the process of comparing two sequences (such as DNA, RNA, or protein) to identify regions of similarity. This process is important for a variety of reasons in bioinformatics, including:

1. **Identifying conserved regions:** It helps identify conserved sequences across species, which can indicate important functional or structural regions of genes or proteins.
2. **Evolutionary analysis:** Alignments are used to infer evolutionary relationships by examining the similarities and differences between sequences.
3. **Mutation analysis:** Pairwise alignment can highlight mutations or differences in sequences, which can be useful for understanding genetic diseases or variations.
4. **Functional predictions:** Alignment allows for the prediction of protein function based on sequence homology.

Pairwise alignment uses scoring schemes to penalize gaps and mismatches, rewarding matches between corresponding nucleotides or amino acids in the two sequences.

b. Using Bio.pairwise2, perform pairwise sequence alignment of two DNA sequences. Print the alignment result and the alignment score.

```
from Bio.Align import PairwiseAligner

# Define two DNA sequences

seq1 = "AGTACACTGGT"

seq2 = "AGTACGCTGGT"

# Create a PairwiseAligner object and perform the alignment

aligner = PairwiseAligner()

alignment = aligner.align(seq1, seq2)

# Print the aligned sequences and the score

print("Aligned Sequences:")

print(alignment[0])

print(f"Alignment Score: {alignment[0].score}")
```

Output Example:

Aligned Sequences:

```
target      0 AGTACA-CTGGT 11
           0 |||||--|||| 12
query       0 AGTAC-GCTGGT 11
```

Alignment Score: 10.0

8 a. What is multiple sequence alignment (MSA), and why is it used in bioinformatics?

Explanation: Multiple Sequence Alignment (MSA) is the alignment of three or more biological sequences (DNA, RNA, or protein) to identify regions of similarity. MSA is essential in bioinformatics because:

- **Evolutionary analysis:** It helps to understand evolutionary relationships among sequences. Sequences that are more similar likely share a common evolutionary origin.
- **Functional prediction:** Conserved sequences across species often indicate regions of biological significance, such as active sites in enzymes or functional domains in proteins.
- **Identifying conserved motifs:** MSA helps in identifying conserved motifs or patterns that are critical for protein function or regulatory elements in DNA.

- **Structure prediction:** In proteins, MSA can help in the identification of conserved secondary structures or folding patterns.

MSA is a crucial step in many bioinformatics workflows, including the study of protein function, structural predictions, and evolutionary studies.

b. Explain the steps to perform a multiple sequence alignment using MUSCLE in Biopython. Write a script to align three sequences of your choice and save the results to a file.

install muscle .exe

https://drive5.com/muscle/downloads_v3.htm

```
import subprocess
```

```
from Bio import AlignIO
```

```
# Use the full path to muscle.exe
```

```
muscle_exe = r"C:\Users\Admin\muscle3.8.31_i86win32.exe"
```

```
try:
```

```
# Run MUSCLE using subprocess
```

```
subprocess.run([muscle_exe, "-in", "input_sequences.fasta", "-out", "aligned_sequences.fasta"], check=True)
```

```
# Read and print the aligned sequences
```

```
alignment = AlignIO.read("aligned_sequences.fasta", "fasta")
```

```
print(alignment)
```

```
except FileNotFoundError:
```

```
print("Error: MUSCLE executable not found. Check the path:", muscle_exe)
```

```
except subprocess.CalledProcessError as e:
```

```
print(f"Error running MUSCLE: {e}")
```

Output Example:

```
>seq1
ATGCGTACGTA
>seq2
ATGCGTACGTC
>seq3
ATGCGTACGAG
```

9. a. What is a phylogenetic tree, and what is its role in bioinformatics?

Explanation: A **phylogenetic tree** is a diagram that represents the evolutionary relationships among different species, genes, or proteins. It illustrates how species or sequences have diverged over time from a common ancestor. In bioinformatics, a phylogenetic tree is used to:

- **Study evolutionary relationships:** By comparing sequences, phylogenetic trees help to infer how closely related different organisms or genes are.
- **Gene/protein function:** Phylogenetic trees can give insights into gene or protein function based on their evolutionary history.
- **Classification of species:** Phylogenetic trees aid in classifying organisms based on shared genetic characteristics.
- **Tracking disease evolution:** Phylogenetics is useful in tracing the evolution of pathogens, such as viruses, which helps in vaccine development and epidemiological studies.

Phylogenetic trees are commonly constructed from sequence data using various algorithms and methods in bioinformatics.

b. Using alignment data, construct a phylogenetic tree and visualize it with Bio.Phylo. Label each branch with the sequence name.

pip install biopython matplotlib

```
from Bio import Phylo, AlignIO

from Bio.Phylo.TreeConstruction import DistanceCalculator, DistanceTreeConstructor

import matplotlib.pyplot as plt

# Load the sequence alignment
alignment = AlignIO.read("aligned_sequences.aln", "clustal")

# Compute distance matrix
calculator = DistanceCalculator("identity")

distance_matrix = calculator.get_distance(alignment)

# Build the phylogenetic tree using UPGMA
constructor = DistanceTreeConstructor()

tree = constructor.upgma(distance_matrix)

# Save tree
Phylo.write(tree, "phylogenetic_tree.nwk", "newick")

# Draw the tree
fig = plt.figure(figsize=(8, 5)) # Set figure size
ax = fig.add_subplot(1, 1, 1) # Ensure only one subplot is used
Phylo.draw(tree, axes=ax) # Draw the tree on the specified axis
plt.show() # Display the tree
```

10 a. What is the Protein Data Bank (PDB), and how is it used to access 3D protein structure data?

Answer: The **Protein Data Bank (PDB)** is a repository of 3D structures of proteins, nucleic acids, and other biomolecules. The data in the PDB is essential for understanding the molecular structure and function of proteins, enzymes, and other biological macromolecules. In bioinformatics, PDB is used for:

- **Studying protein structure:** Researchers access 3D structures to understand how a protein functions, interacts with other molecules, and folds into its active shape.
- **Drug design:** Structural data from the PDB is used to design drugs that can bind to specific proteins, such as enzymes or receptors, by targeting their active sites.
- **Structural bioinformatics:** PDB files provide valuable information for predicting the 3D structure of proteins based on their sequence.

b. Fetch a protein structure from the Protein Data Bank (PDB) using Bio.PDB and visualize the 3D structure of the protein. Perform basic manipulations like selecting a region or displaying specific chains.

pip install biopython matplotlib numpy

```
from Bio.PDB import PDBList, PDBParser
```

```
import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
import numpy as np
```

```
# Step 1: Download a PDB structure (if not already downloaded)
```

```
pdb_id = "1A3N" # Replace with any valid PDB ID
```

```
pdb_filename = f"{pdb_id}.pdb"
```

```
pdbl = PDBList()
```

```
pdbl.retrieve_pdb_file(pdb_id, pdir=".", file_format="pdb")
```

```
# Step 2: Parse the PDB file
```

```
parser = PDBParser(QUIET=True)
```

```
structure = parser.get_structure(pdb_id, f"pdb{pdb_id}.ent")
```

```
# Step 3: Extract atomic coordinates
```

```
atoms = []

for model in structure:

    for chain in model:

        for residue in chain:

            for atom in residue:

                atoms.append(atom.coord)

atoms = np.array(atoms) # Convert list to NumPy array for easy plotting

# Step 4: Visualize the 3D structure using Matplotlib

fig = plt.figure(figsize=(8, 6))

ax = fig.add_subplot(111, projection="3d")

ax.scatter(atoms[:, 0], atoms[:, 1], atoms[:, 2], c="blue", marker="o", s=10)

ax.set_title(f"3D Structure of {pdb_id}")

ax.set_xlabel("X-axis")

ax.set_ylabel("Y-axis")

ax.set_zlabel("Z-axis")

plt.show()
```