What is Biopython?

Biopython is a set of freely available **Python tools for computational biology and bioinformatics**.
It allows scientists and developers to **work with biological data** like DNA, RNA, and protein sequences, as well as biological databases.

Key Features:

- Sequence analysis (DNA, RNA, Protein)
- Reading/writing sequence files (FASTA, GenBank, etc.)
- Accessing online biological databases (NCBI, UniProt)
- Performing bioinformatics algorithms (alignment, translation, transcription)
- Working with phylogenetic trees
- Parsing biological data formats

Use in real life:
If you want to automate tasks like fetching gene sequences, analyzing protein structures, or performing sequence alignments, Biopython makes it easy.

Why use Biopython?

- **Ease of use:** Python syntax is simple, so bioinformatics tasks become easier.
- **Automation:** Fetch sequences or run analyses programmatically instead of manually.
- **Integration:** Works with Python's scientific libraries like NumPy, Pandas, Matplotlib.
- **Community support:** Widely used in research, with lots of tutorials and documentation.

Example Tasks:

- Extract a gene sequence from GenBank.
- Translate DNA to Protein.
- Perform multiple sequence alignment.
- Visualize phylogenetic trees.

How to Install Biopython

You can install Biopython using **pip**:

```
pip install biopython
```

Check if installed correctly:

```
import Bio
print(Bio.__version__)
```

If it prints a version number, it's installed successfully.

What is DNA and RNA?

DNA (Deoxyribonucleic Acid)

- DNA is the **genetic material** in almost all living organisms.
- It carries instructions to make proteins and to control cell activities.
- Structure: **Double helix** made of **nucleotides**.
- Each nucleotide has **a base, sugar (deoxyribose), and phosphate**.

Bases in DNA:

- **A** = Adenine
- **T** = Thymine
- **G** = Guanine
- **C** = Cytosine

Base pairing rules:

- A pairs with T → A-T
- G pairs with C → G-C

RNA (Ribonucleic Acid)

- RNA is a **single-stranded molecule**.
- Helps **carry genetic code from DNA to make proteins**.
- Structure: Sugar is **ribose**, base **U (Uracil)** instead of T.

Bases in RNA:

- A → U (Uracil)
- G → C
- C → G
- T → A (from DNA)

What is ATGC?

- ATGC are **the letters representing the DNA bases**.
- A DNA sequence is just a **string of these letters**, e.g., `"ATGCGTAC"`.
- Patterns in DNA are called **motifs** or **genes**, which may encode proteins.

Basic Operations in Bioinformatics

- **Complement:** Replace each base with its pair: A↔T, G↔C
- **Reverse Complement:** Reverse the DNA sequence, then take complement
- **Transcription:** Convert DNA → RNA (T becomes U)
- **Translation:** Convert RNA sequence → Protein sequence (sequence of amino acids)

Python Code Example

```
from Bio.Seq import Seq

# Create DNA sequence
dna_seq = Seq("ATGCGTAC")

# Print original DNA
print("DNA sequence:", dna_seq)

# Complement of DNA
print("Complement:", dna_seq.complement())

# Reverse complement
print("Reverse complement:", dna_seq.reverse_complement())

# Transcription (DNA to RNA)
print("Transcription (DNA to RNA):", dna_seq.transcribe())

# Translation (RNA to Protein)
print("Translation (RNA to Protein):", dna_seq.translate())
```

Step by Step Output Explanation

Suppose DNA = `ATGCGTAC`

Operation	Result	Meaning
DNA sequence	ATGCGTAC	Original sequence
Complement	TACGCATG	Pair each base (A↔T, G↔C)
Reverse complement	GTACGCAT	Complement, then reverse
Transcription (DNA → RNA)	AUGCGUAC	Replace T with U
Translation (RNA → Protein)	M R Y	Converts codons (3 letters) to amino acids

What is a FASTA file?

- **FASTA format** is a **text-based format** used to store **nucleotide (DNA/RNA) or protein sequences**.
- Each sequence has **a header** (starts with > symbol) and the **sequence itself** in the next lines.

Example of a FASTA file:

```
>Sequence_1
ATGCGTACGTTAG
>Sequence_2
CGTAGCTAGCTA
>Sequence_3
ATGGGCTTAA
```

- > indicates the **sequence name or ID**
- Next lines contain the **sequence letters** (A, T, G, C for DNA; A, U, G, C for RNA; 20 letters for proteins).

How to Create a FASTA file

You can create it **manually** in a text editor or **programmatically using Python**:

```
# Create a FASTA file with multiple sequences
sequences = {
    "Sequence_1": "ATGCGTACGTTAG",
    "Sequence_2": "CGTAGCTAGCTA",
    "Sequence_3": "ATGGGCTTAA"
}

with open("example.fasta", "w") as file:
    for name, seq in sequences.items():
        file.write(f">{name}\n{seq}\n")
```

This will create `example.fasta` in the same folder.

How to Read a FASTA file using Biopython

We use the module **`SeqIO`** from Biopython.

```
from Bio import SeqIO

# Read the FASTA file
for record in SeqIO.parse("example.fasta", "fasta"):
    print("ID:", record.id)           # Sequence name
    print("Sequence:", record.seq)    # Sequence letters
    print("Length:", len(record))     # Sequence length
    print("---")
```

Output:

```
ID: Sequence_1
Sequence: ATGCGTACGTTAG
Length: 13
---
ID: Sequence_2
Sequence: CGTAGCTAGCTA
Length: 12
---
ID: Sequence_3
Sequence: ATGGGCTTAA
Length: 10
---
```

Working with Different Sequences

You can store **DNA, RNA, or Protein sequences** in the same FASTA file. Example:

```
>DNA_seq
ATGCGTACGTTAG
>RNA_seq
AUGCGUACGUUAG
>Protein_seq
MRTKQ
```

Python code will still read them the same way:

```
for record in SeqIO.parse("example.fasta", "fasta"):
    print("ID:", record.id)
    print("Sequence:", record.seq)
```

Summary

Feature	FASTA Example / Python Function
File format	Text file with > header
Sequence types	DNA, RNA, Protein
Create FASTA	Python `open()` + write
Read FASTA	`from Bio import SeqIO`
Access sequence	`record.seq`
Access ID	`record.id`
Sequence length	`len(record)`